Anais do XIV SEMINCO

Seminário de Computação

19 a 21 de Outubro de 2005

FURB - Campus IV - Blumenau/SC

Promoção

Universidade Regional de Blumenau - FURB
Pró-Reitoria de Extensão e Relações Comunitárias - PROERC
Centro de Ciências Exatas e Naturais - CCEN
Departamento de Sistemas e Computação - DSC
Centro Acadêmico Livre de Computação - CALCOMP

Comissão Organizadora

Prof. Everaldo Artur Grahl (Coordenador)
Prof. Alexander Roberto Valdameri
Prof. Antônio Carlos Tavares
Prof. Jomi Fred Hübner
Profa. Joyce Martins
Prof. José Roque Voltolilni da Silva

Ficha Catalográfica Elaborada pela Biblioteca da FURB

Seminário de Computação (14.: 2005 : Blumenau, SC)

Anais do XIV SEMINCO / promoção Universidade Regional de Blumenau, Departamento de Sistemas e Computação; Everaldo Artur Grahl, Jomi Fred Hübner (coordenador). - Blumenau, O Departamento, 2005. 200 p. : il.

1. Computação - Congressos. I. Grahl, Everaldo Artur. II. Universidade Regional de Blumenau. Departamento de Sistemas e Computação. III. Título.

CDD 004

Apresentação

A Universidade Regional de Blumenau - FURB, através do Departamento de Sistemas e Computação, realiza o XIV Seminário de Computação (SEMINCO) e a II Mostra de Software entre os dias 19 e 21 de outubro de 2005.

Este ano tivemos a submissão de 47 artigos provenientes de várias instituições e empresas do país, sendo que destes foram aprovados 14 artigos das seguintes universidades e empresas: UNIPLAC, FURB, UNISINOS, USP, ITA; Total, Ápice, Senior. A organização dos artigos nestes anais é feita conforme as seguintes áreas de conhecimento: Banco de Dados, Computação Gráfica, Engenharia de Software, Informática na Educação, Inteligência Artificial, Integração Hardware/Software e Rede de Computadores.

Agradecemos a todos os envolvidos na organização do evento, bem como a Comissão de Avaliação Interinstitucional que não mediu esforços em avaliar os diversos artigos submetidos à chamada de trabalhos. Esperamos que nos três dias de realização do evento as expectativas dos participantes sejam atendidas e tenhamos um grande evento. Até o ano que vem!

Comissão Organizadora

Agradecimentos

Sociedade Brasileira de Computação - SBC
Comissão de Avaliação Interinstitucional
Projeto Acredito - FURB

Comissão de Avaliação Interinstitucional

Artigos Selecionados

Banco de Dados

Computação Gráfica

Engenharia de Software

Informática na Educação

Inteligência Artificial

Integração Hardware/Software

Rede de Computadores

Mostra de Software

Uso de SIG como Suporte à Decisão em Programa de Assistência Odontológica

Angelo Augusto Frozza (UNIPLAC)
frozza@uniplac.net

Rogeria Ramos Monteiro (UNIPLAC)
rogeria@uniplac.net

João Alexandre Cordova de Sousa (UNIPLAC)
jaosousa@uniplac.net

Renato Valiati (UNIPLAC)
valiati@uniplac.net

Resumo. Este artigo descreve as atividades realizadas pelo Grupo de Pesquisa em Sistemas de Informações Geográficas (GpSIG) da UNIPLAC. O enfoque dos trabalhos é dado à criação de ferramentas para auxiliar na tomada de decisões nos projetos de extensão da instituição, mais precisamente, o Programa de Assistência Odontológica – PROASOD. A tática adotada é a disponibilização de uma aplicação *Web* (SIGOdonto) que permita a execução de consultas diversas e visualização de suas respostas sobre o mapa da região de interesse e em formato de relatório, a fim de facilitar a elaboração de planos estratégicos de cunho administrativo. Tais consultas podem ser feitas sobre atributos específicos ou interagindo com o próprio mapa. O SIGOdonto foi elaborado com tecnologias de código aberto, tais como *MapServer*, *Postgre*SQL/*Post*GIS, *Linux* e PHP.

Palavras-chave: Sistemas de Informações Geográficas; tomada de decisão; assistência odontológica; *MapServer*.

1 Introdução

"O conceito fundamental dos vários modelos de tomada de decisão é o de *racionalidade*. De acordo com este princípio, indivíduos e organizações seguem um comportamento de escolha entre alternativas, baseado em critérios objetivos de julgamento, cujo fundamento será satisfazer um nível pré-estabelecido de aspirações" (DPI/INPE, 2005). O processo decisório consiste em realizar uma escolha a partir de alternativas. Com base nesta visão, é possível interpretar o processo de manipulação de dados em um Sistema de Informações Geográficas (SIG) como uma forma de produzir diferentes hipóteses sobre um tema de estudo (DPI/INPE, 2005).

Diversos autores apresentam conceitos diferentes para SIGs, contudo, todos indicam a variedade de operações e visões possíveis desta tecnologia. Sendo assim, uma definição apropriada de SIG é "um sistema de suporte à decisão que integra dados referenciados espacialmente num ambiente de respostas a problemas" (CÂMARA, 1995). Estes sistemas manipulam dados de diversas fontes e formatos, dentro de um ambiente computacional ágil e capaz de integrar informações espaciais temáticas e gerar novas informações, derivadas dos dados originais (CARVALHO, 1999).

Sistemas de Informações Geográficas separam a informação em diferentes camadas temáticas (*layers*) e as armazenam independentemente, permitindo trabalhar com elas de modo rápido e simples. Desta forma, os usuários têm a oportunidade de relacionar a informação existente através da posição e formato (polígono ou ponto) dos objetos, com o fim de gerar nova informação.

"O uso de técnicas de geoprocessamento na área de saúde permite a análise da distribuição espacial de agravos, de problemas ambientais relacionados e a avaliação das redes de atenção à saúde" (MAGALHÃES *et al*, 2003).

Conhecendo-se onde se localizam as comunidades com maior necessidade de serviços a serem prestados na área odontológica, bem como, quais os principais tipos de atendimento necessários, pode-se desenvolver melhores políticas de prestação de serviço, tanto por parte da UNIPLAC como do Município de Lages. Os resultados deste tipo de análise podem ser quantificados de forma mais adequada se forem relacionados os custos de execução de cada atendimento com a infra-estrutura disponível na UNIPLAC e nos centros de atendimento mantidos pela Prefeitura, possibilitando deste modo, a otimização do uso dos mesmos.

Desta forma, em 2004 os cursos de Sistemas de Informação e Odontologia motivaram-se para criar o Grupo de Pesquisa em Sistemas de Informações Geográficas (GpSIG), tendo como primeiro produto de pesquisa desenvolvida o "Sistema de Informação Geográfica aplicado ao Programa de Assistência Odontológica da UNIPLAC", ou SIGOdonto.

2 A proposta do SIGOdonto

A Universidade do Planalto Catarinense possui, desde 2002, um Programa de Assistência Odontológica (PROASOD) praticado através do curso de Odontologia. Os serviços totalizam, desde então, mais de 12.500 atendimentos, cujos registros são feitos com o auxílio de dois *softwares*: Sistema Informatizado de Controle Acadêmico (SICA) e Sistema Integrado de Atendimento Comunitário (SIAC). Os relatórios atuais são fornecidos somente no formato textual/tabular (Quadro 1), dificultando o processo decisório, pois não é possível ter noção exata da dimensão espacial do grande volume de dados obtido pelos atendimentos realizados no Programa.

Quadro 1: Exemplo de relatório tabular dos atendimentos no PROASOD (*Dados fictícios).

NOME (*)	NASC.	ENDEREÇO	SERVIÇOS REALIZADOS	UNIT.	TOTAL
Maria ...	10/08/1987	Independência nº 78, Bairro Gethal	Dentística Básica Consulta	30,00 32,00	30,00 32,00
Pedro ...	25/02/1987	Aníbal de Athayde nº 620, Bairro Tributo	Cirurgia II	120,00	120,00
João ...	20/05/1953	Alexandre Gonzatto nº 55. BairroFerrovia	Cirurgia II	120,00	240,00

Neste sentido, Sistemas de Informações Geográficas vêm ao encontro das necessidades dos gestores do PROASOD, como uma ferramenta vital para conhecer e compreender precisamente o público alvo de seus trabalhos. A aplicação de tal tecnologia, pode constatar regiões onde, por exemplo, a população seja menos favorecida sócio-economicamente e tenha baixa demanda no Programa. A partir disso, os coordenadores do PROASOD podem elaborar planos estratégicos, em conjunto com o Poder Público Municipal, para agir em tais locais, além de apurar as reais causas de tal fato.

A estratégia adotada pela equipe do GpSIG neste projeto foi disponibilizar na Internet uma aplicação para realizar consultas, tanto por atributos quanto por apontamentos e obter as respectivas respostas no mapa da cidade de Lages, divido por bairros, em conjunto com relatórios pré-formatados. A tecnologia *WebMapping* utilizada foi o *MapServer*, justamente por viabilizar o desenvolvimento de aplicações em ambiente *Web* que propiciam ao usuário a capacidade de visualizar e interagir com mapas e atributos presentes em uma ou mais fontes de dados. Desta forma, o usuário é beneficiado com interfaces simples e práticas, aliadas à facilidade de

interpretação de dados visuais (mapas, gráficos etc.). O SIGOdonto está disponível através do endereço http://inf.uniplac.net/gpsig.

2.1 Estrutura da aplicação *Web* SIGOdonto

A arquitetura do SIGOdonto é apresentada na Figura 1. Seu funcionamento baseia-se em três arquivos principais que, juntos, disponibilizam alternativas de consulta e apresentam respostas aos gestores do PROASOD, formando uma base de conhecimento precisa para seu processo decisório:

1. `init.php`: página de acesso inicial, onde o usuário faz suas opções de consulta e, de forma oculta, contém as variáveis de inicialização do *MapServer*;
2. `template.htm`: página que apresenta o resultado da consulta escolhida no `init.php`, ou seja, mostra o mapa colorido de acordo com as ocorrências relativas à pergunta escolhida e, em alguns casos, também mostra informações textuais correlacionadas;
3. `mapalages.map`: arquivo com a definição das *layers* (camadas) de informação que podem ser sobrepostas, conforme a consulta de interesse, gerando novas informações.
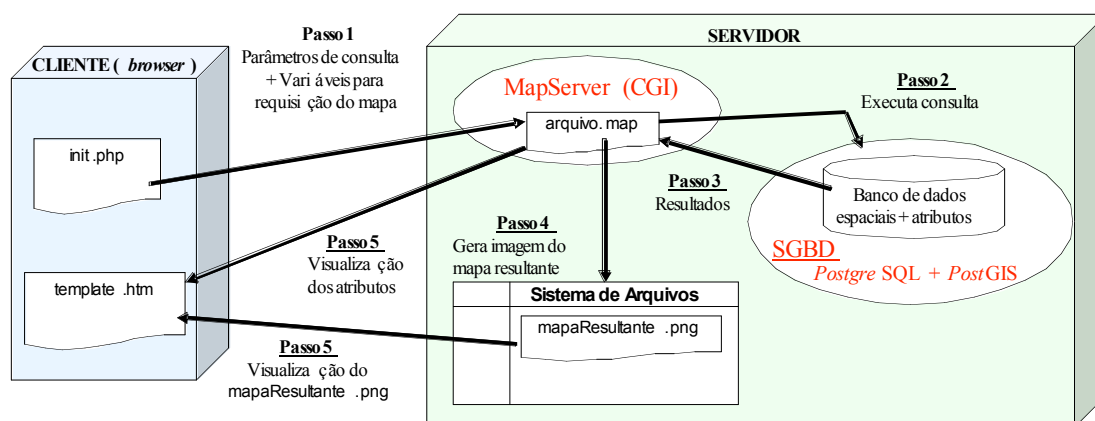
Figura 1: Estrutura do *site* SIGOdonto.

O SIGOdonto foi hospedado em um servidor com processador *Intel Pentium* 800 Mhz, no qual foram instalados os *softwares* necessários: sistema operacional Conectiva *Linux* 9; banco de dados *Postgre*SQL 8.0 com extensão geográfica *Post*GIS 1.0; servidor *web Apache* 2.0: pacote aplicativos de *webmapping* FGS, contendo o servidor de mapas *Mapserver* 4.0. A implementação utilizou ainda linguagens de programação padrões da *web*,como HTML 4.0, CSS 1.0 e *JavaScript*, além do PHP para acesso a banco de dados e partes dinâmicas do *site*. A aplicação está disponível a partir do endereço `http://inf.uniplac.net/gpsig`.

2.2 O arquivo de configuração

Um dos pontos mais críticos do sistema é a montagem do arquivo de configuração das camadas de informação do mapa: `mapalages.map`. Sua estrutura está representada no quadro 2, acompanhada de comentários para auxiliar o entendimento. As informações de configuração dividem-se em seções, conforme segue: linhas 02 a 10 – configurações iniciais; linhas 11 a 18 – configurações para o servidor *web*; linhas 19 a 27 – configurações de legenda; linhas 28 a 41 – configurações da barra de escala; linhas 43 a 47 – configurações para os resultados das consultas; linhas 49 a 54 – configurações de cada camada de informações.

Quadro 2: Fragmento do arquivo mapalages.map.

```
01   MAP                          #Início das configurações
02     NAME MapaLages             #Identificador do arquivo de configuração
03     STATUS ON
04     EXTENT -8.5 -9.5 7 9.5     #Coordenadas geográficas do mapa criado.
05     SIZE 680 580               #Tamanho em pixels das imagens geradas.
06     #localização dos simbolos e fontes utilizados nos mapas.
07     SYMBOLSET "../projeto1/simbolos.sym"
08     FONTSET "../projeto1/fontes.txt"
09     IMAGETYPE PNG              #Formato da imagem a ser gerada.
10     IMAGECOLOR 255 255 255     #Cor padrão do mapa
11     WEB
12       #Template para a tela de apresentação do mapa.
13       TEMPLATE 'templateComLegendas.htm'
14       #Caminho onde são armazenados os mapas gerados.
15       IMAGEPATH '/ms4w/Apache/htdocs/projeto1/projeto1/tmp/ms_tmp/'
16       #URL onde são armazenados os mapas gerados.
17       IMAGEURL  '/projeto1/projeto1/tmp/ms_tmp/'
18     END
19     LEGEND              #Configurações da legenda.
20       KEYSIZE 18 12     #Tamanho da legenda.
21       LABEL             #Formatações de escrita dentro da legenda.
22         TYPE BITMAP     #Tipo da fonte.
23         SIZE MEDIUM     #Tamanho da fonte.
24         COLOR 0 0 89    #cor da fonte.
25       END
26       STATUS on         #Se a legenda permanece visível ou não.
27     END
28     SCALEBAR                   #Configurações da escala
29       IMAGECOLOR 255 255 255   #Cor da escala
30       LABEL
31         COLOR 0 0 0            #Cor do texto na escala
32         SIZE TINY              #Tamanho da fonte do texto na escala
33       END
34       OUTLINECOLOR 200 0 0     #Cor das linhas externas da escala
35       STYLE 0                  #Tipo de escala a ser utilizada
36       SIZE 150 10              #Tamanho da escala a ser gerada
37       COLOR 0 0 0              #Cor
38       UNITS METERS             #Unidade - metros
39       INTERVALS 2              #Espaço de entre os intervalos na escala
40       TRANSPARENT False        #Transparência da escala
41       STATUS on #EMBED         #Se escala é gerada dentro ou fora do mapa
42     END
43     QUERYMAP            #Mapeamento dos resultados de uma consulta
44       STATUS on         #ON - se for para desenhar o mapa, OFF - se não
45       STYLE selected    #Mostra somente os dados escolhidos
46       COLOR 255 255 0   #Cor para os resultados da consulta
47     END
48     #----- Definições das Camadas de Informação ------------
49     LAYER                      #inicio da primeira camada
50       ...
51     END
52     LAYER                             #inicio de uma segunda camada
53       …
54     END
55   END
```

Cada camada de informação é configurada em uma *layer* distinta no arquivo `mapalages.map` (quadro 3). Basicamente, cada *layer* precisa conhecer: o tipo de conexão com o banco de dados; um identificador único; o script de consulta ao banco de dados, incluindo os relacionamentos com as informações espaciais; o tipo de geometria de cada objeto espacial (ponto, linha ou polígono). Se a camada apresenta a informação em classes de valores como, por exemplo, faixas de idade, cada classe de valor precisa ter sua própria configuração (por exemplo, linhas 22 a 27).

Quadro 3: Fragmento do arquivo mapalages.map – configuração de uma camada.

```
01    LAYER
02      NAME "num_habitantes"        #Identificador da camada
03      #Definições para a conexão com o banco de dados remoto
04      CONNECTIONTYPE postgis       #Tipo de conexão com BD utilizada
05      CONNECTION "user=xxx dbname=xxxxxx host=localhost"
06      #Script de consulta no banco de dados
07      DATA "the_geom from (select bairroslages.gid as oid,
08            bairroslages.the_geom as the_geom,
09            bairros.numhabitantes as habitantes
10            from bairroslages left join bairros on
11            bairroslages.gid = bairros.codbairro)
12            as nomesbairros using unique oid using srid=-1"
13      STATUS off            #A camada só aparece quando for solicitada
14      TYPE POLYGON          #Geometria dos objetos consultados

15      #Define a sensibilidade para pesquisas via mouse
16      TOLERANCEUNITS pixels
17      TOLERANCE 5

18      #Uma camada pode possuir classes de informação que são
19      #identificadas por configurações distintas (por exemplo: a cor)
20      CLASS                        #Definições de aparência da classe
21        NAME "Até 500 habitantes"        #Nome da classe
22        EXPRESSION ([habitantes] <= 500 ) #Seleção dos dados
23        COLOR 167 198 157                #Cor da imagem gerada
24        OUTLINECOLOR 0 0 0               #Contorno dos polígonos
25      END
26      CLASS
27        NAME "501 a 1000 habitantes"
28        EXPRESSION ([habitantes] >= 501 and [habitantes]<= 1000 )
29        COLOR 196 189 67
30        OUTLINECOLOR 0 0 0
31      END
32      CLASS
33        NAME "1001 a 2500 habitantes"
34        ...
35      END
36      CLASS
37        NAME "2501 a 5000 habitantes"
38        ...
39      END
40      CLASS
41        NAME "5001 a 10000 habitantes"
42        ...
43      END
44    END
```

2.3 Tratamento dos dados espaciais e atributos correlacionados

O Sistema Gerenciador de Banco de Dados (SGBD) escolhido para este projeto foi o *Postgre*SQL, por possuir uma extensão dedicada a dados espaciais, denominada *Post*GIS. O mapa da cidade de Lages, dividido por bairros, precisou ser manipulado em aplicativos de desenho vetorial a fim de gerar um arquivo no formato `.shp` (*shapefile*). Optou-se por este formato por ser um padrão de ampla aceitação para transferência de dados geográficos. A importação do mesmo para *Post*GIS foi realizada através da ferramenta `shp2pgsql` (que acompanha o *Post*GIS) usando o seguinte comando:

```
shp2pgsql arquivoMapa.shp nomeBd tabelaDestino > dadosMapa.sql
```

Este comando gera as instruções que convertem o `arquivoMapa.shp` em uma tabela suportada pelo *Post*GIS, cujo nome é `tabelaDestino` e que está no banco de dados `nomeBd`. Como resultado final, grava as instruções na forma de *script* SQL que é armazenado no arquivo `dadosMapa.sql`. Após este procedimento, o arquivo gerado com as instruções SQL é usado para inclusão dos dados geográficos no *Postgre*SQL.

As informações do Programa de Assistência Odontológica que são relevantes a este estudo foram devidamente modeladas conforme diagrama apresentado na Figura 2, no qual aparece a ligação entre o modelo de dados e as informações espaciais, representados pelo relacionamento 1:1 entre as tabelas `MapaBairrosLages` e `Bairros`.
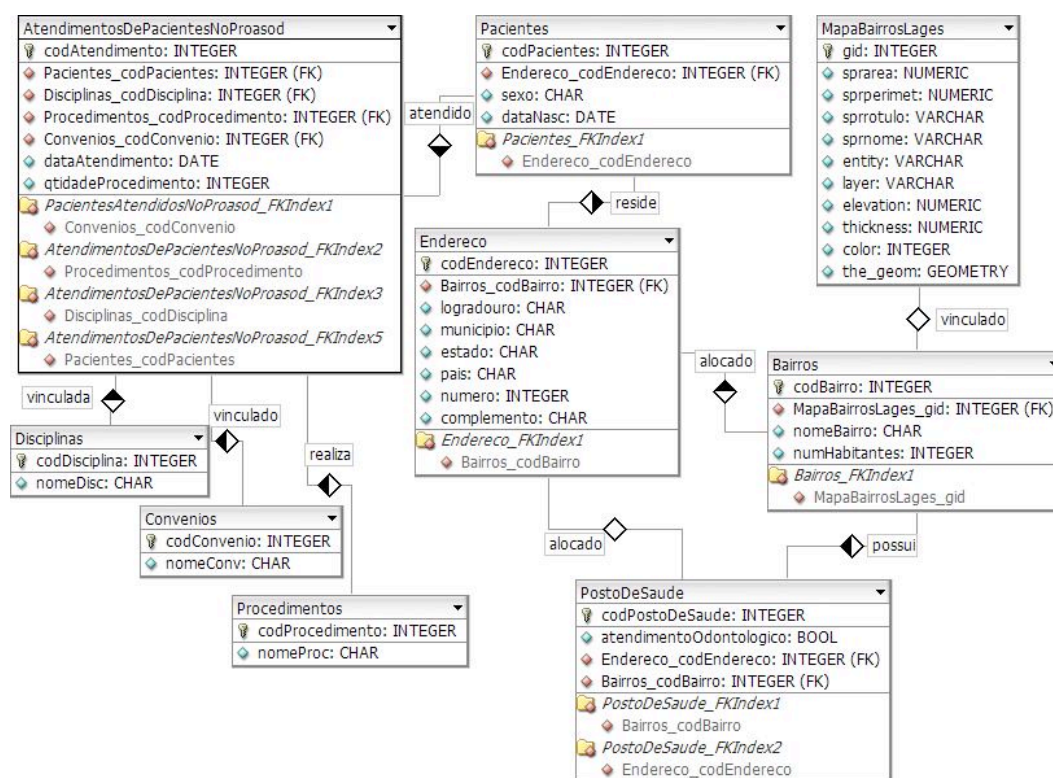
Figura 2: Modelo de dados do SIGOdonto.

3 Usando o SIGOdonto para a tomada de decisão

Um SIG deve possuir duas formas de consulta geográfica: espacial (que pode ser gráfica) e por atributos (geralmente pela seleção de parâmetros). A primeira, normalmente é utilizada para restringir uma determinada área ou região de interesse. A outra é utilizada para selecionar e analisar os geo-objetos que satisfazem às condições solicitadas pelo usuário. Através da consulta *espacial* pode-se limitar não somente o local de interesse, mas também diminuir o número de dados com o qual se opera uma consulta por *atributos*. Portanto, a utilização racional destas duas formas de consulta pode propiciar um bom desempenho ao sistema, independentemente da quantidade de dados disponíveis no banco de dados e que devem ser manuseados (HARA 1997).

Conforme relatado na sessão anterior, a primeira página do *site* (`init.php`) contém as opções de consulta, além de instruções de uso do SIGOdonto (Figura 3).

Figura 3: Tela inicial do SIGOdonto.

Foram disponibilizados cinco tipos de consulta:

1. *Por atributo*: o usuário seleciona o bairro a partir de uma lista. O sistema apresenta o mapa da cidade, com os bairros representados por polígonos, destacando o bairro selecionado e informações correlatas (Figura 4);

Esta consulta tem por objetivo possibilitar a visualização de informações sobre um bairro específico. A tela de resultado tem a seguinte estrutura: à esquerda apresenta-se o mapa gerado dinamicamente; à direita aparece, de cima para baixo, a identificação do Grupo de Pesquisa e alguns atributos relacionados com a consulta ou a legenda para identificação dos elementos no

mapa. No canto inferior esquerdo há um botão que leva o usuário à tela principal. Esta mesma estrutura é usada nas demais telas de consulta.
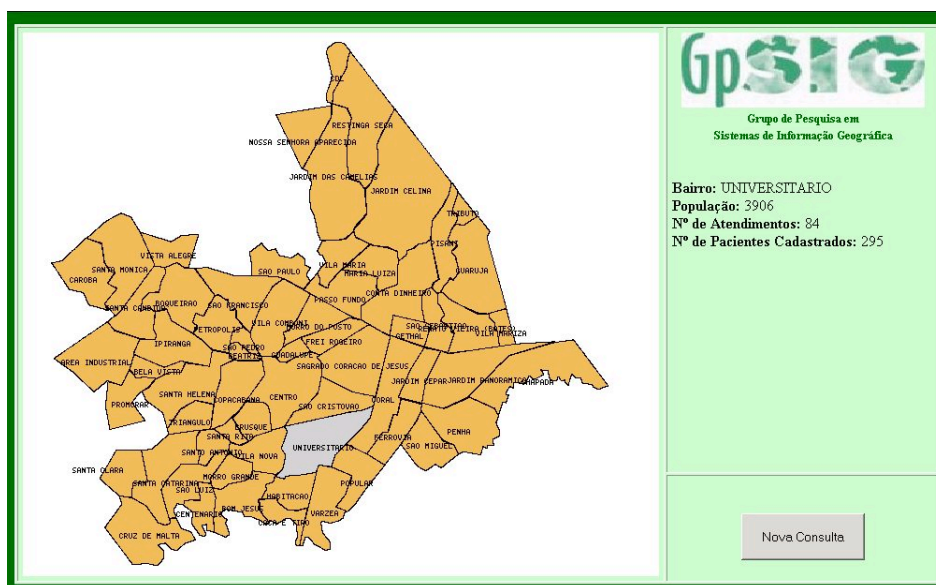
Figura 4: Resposta da consulta por um Bairro.

2. *Sobreposição de layers* combinada *com consulta por atributos*: neste tipo de consulta, o sistema combina diversas informações em um com único mapa (representação por polígono: bairros; por pontos: localização de postos de saúde; por *labels*: anotações gerais, como os nomes dos bairros). Entre as opções de consultas estão: Relação de Atendimentos *versus* Tipos de Procedimentos, Relação de Atendimentos *versus* Tipos de Convênio e Relação de Atendimentos *versus* Disciplinas do curso de Odontologia associadas aos Atendimentos (Figura 5);

Por meio desta consulta, os gestores do PROASOD podem identificar, por exemplo, de onde vem os pacientes que se submetem a tratamento de canal (endodontia) e se os mesmos possuem algum tipo de convênio. Com estes dados, pode-se identificar alguma concentração de determinado tipo de atendimento em certos bairros. A partir de então, cabe fazer um trabalho de investigação para levantar o porque desta concentração e, por conseqüência, podem ser identificadas novos dados para alimentar o sistema, os quais podem ser de ordem econômico-social, ambiental etc.

3. *Layers simples*: a saída corresponde a um mapa da cidade com os bairros representados por polígonos e a apresentação de um único tipo de informação. Deste modo, as consultas disponíveis são: População por Bairro, Número de Habitantes Cadastrados no PROASOD por Bairro e Número de Habitantes Atendidos por Bairro (Figura 6);

Este tipo de consulta é útil para avaliar cada classe de informação separadamente como, por exemplo, quais os bairros mais populosos e de onde vem os pacientes cadastrados no Programa. Uma das informações mais importantes obtidas por meio desta consulta foi a indicação de que a

maior parte dos pacientes atendidos no programa são oriundos dos bairros vizinhos ao bairro onde localiza-se a Universidade, os quais nem sempre são os menos favorecidos sócio-economicamente. Desta forma, ficam em aberto outras questões que merecem investigação como, por exemplo, que fatores levam ao baixo número de pacientes cadastrados e atendidos no Programa, oriundos dos bairros menos favorecidos sócio-economicamente e mais distantes da Universidade.
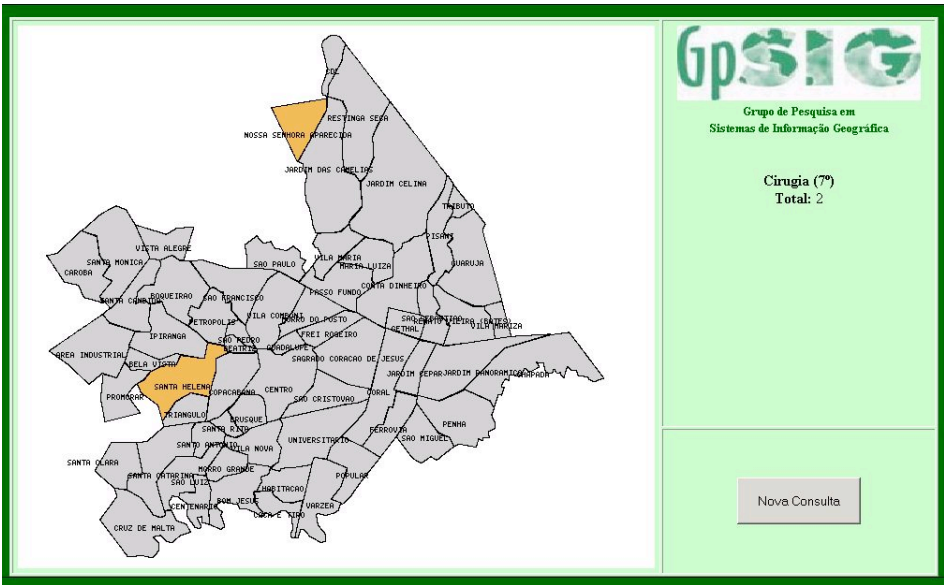
Figura 5: Resultado da consulta Relação de Atendimentos X Disciplina.
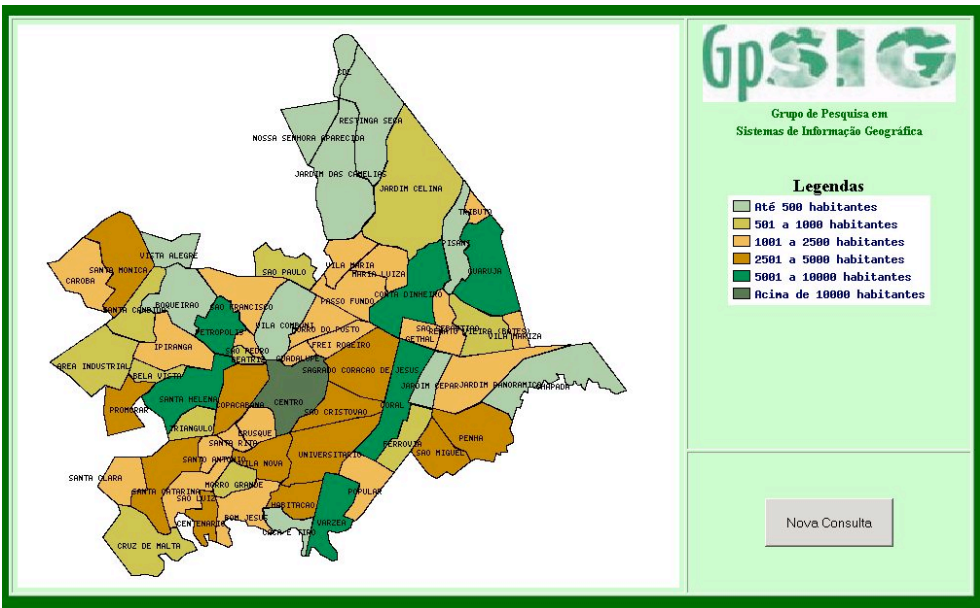
Figura 6: Resultado da consulta População por Bairro.

4. *Sobreposição de layers*: usa a representação por polígonos e por pontos. As consultas possíveis são: Relação de Atendimentos *versus* População por Bairro, Relação de Habitantes Cadastrados no PROASOD *versus* População por Bairro e Relação de Habitantes Atendidos *versus* Habitantes Cadastrados no PROASOD (Figura 7);

Este tipo de consulta possibilita fazer o levantamento do percentual da população de cada bairro que está cadastrada ou é atendida no PROASOD. A partir dela, percebe-se que esta relação não é homogênea, onde em alguns bairros o Programa atinge um percentual maior de habitantes do que em outros.
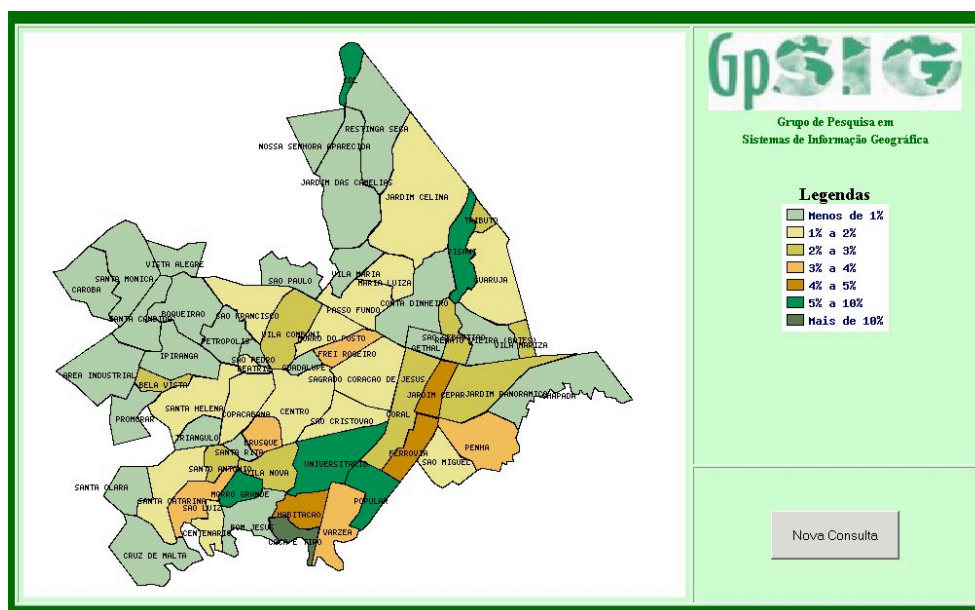
Figura 7: Resultado da consulta Relação de Habitantes Cadastrados no PROASOD X População por Bairro.

5. *Por apontamento no mapa*: um mapa dos bairros da cidade é mostrado (representação por polígonos). O usuário seleciona com o *mouse* o bairro de interesse, o qual é destacado e informações estatísticas referentes ao bairro são apresentadas (Figura 8).

Este tipo de consulta permite que, ao visualizar um mapa, o usuário possa escolher com o *mouse* um bairro específico e obter informações mais detalhadas sobre este bairro, eliminando a necessidade de voltar à tela principal.

Soma-se aos tipos de consulta acima citados, a possibilidade de visualizar os bairros que têm Postos de Saúde mantidos pelo Poder Municipal, qual a localização destes Postos de Saúde e se os mesmos oferecem ou não atendimento odontológico à população. Futuramente, pretende-se adicionar ao SIGOdonto informações oriundas dos atendimentos odontológicos realizados nestes Postos de Saúde.
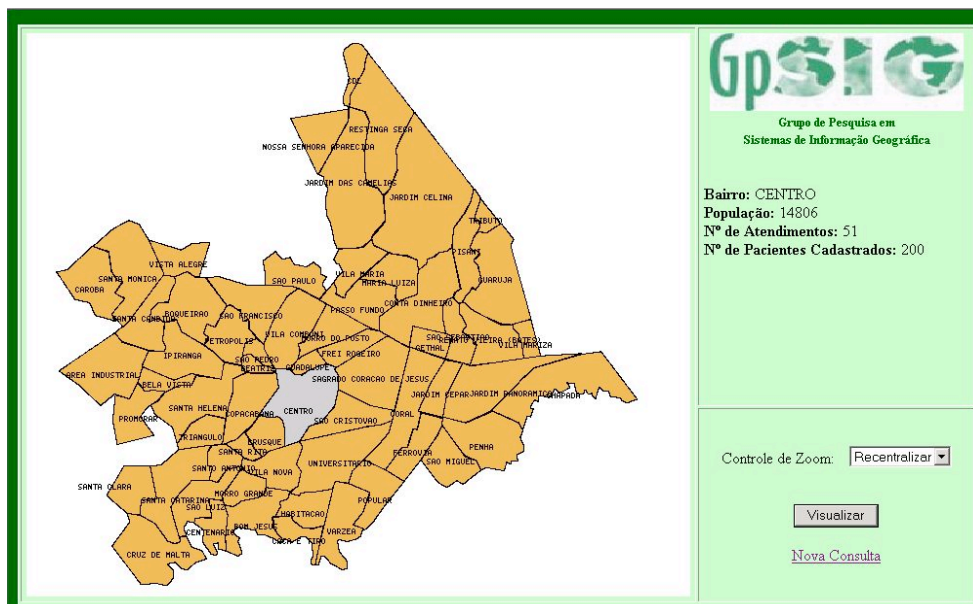
Figura 8: Resultado da seleção direta de um Bairro no mapa.

4 Considerações Finais

O projeto SIGOdonto veio ao encontro das necessidades dos gestores do PROASOD, no sentido de facilitar o conhecimento da abrangência dos serviços prestados e apoiar decisões estratégicas a fim de melhorar o Programa já implantado. O fato de ter as informações de interesse apresentadas em um formato visual, aliado à interatividade oferecida, representa um ganho altamente significativo na administração do Programa.

É bem verdade que para chegar a este resultado foi preciso dedicar um período para a identificação da tecnologia de *WebMapping* mais adequada ao propósito estabelecido e, também, para o aprendizado da mesma. Neste sentido, optou-se pelo *MapServer*, devido ao fato de ser um *software* livre e à existência de uma comunidade de usuários expressiva, cujo auxílio foi fundamental em muitos momentos críticos.

Uma das metas alcançadas foi a estruturação de uma arquitetura Cliente-Servidor em ambiente *Web*, onde os usuários realizam consultas no módulo Cliente, tanto por atributos como por apontamentos no próprio mapa. Estas requisições, por sua vez, são enviadas ao módulo Servidor, que realiza o processamento da solicitação e retorna ao Cliente o mapa resultante, juntamente com os respectivos atributos.

Com relação às informações até o momento disponíveis, pode-se citar a descoberta de que o maior número de pacientes atendidos encontra-se nos bairros ao arredor da Universidade, não necessariamente sendo as regiões menos favorecidas sócio-economicamente. Esta informação está sendo analisada para identificar por que outras regiões têm menos pacientes atendidos.

Ainda restam algumas atividades a serem realizadas, como o acompanhamento do uso da ferramenta pelos gestores do PROASOD, a fim de se avaliar a importância das consultas disponibilizadas e criar novas consultas que auxiliem em situações por hora não identificadas.

Outro campo de expansão se refere à inclusão de novos atributos provenientes do Poder Público, por meio de convênio entre a Prefeitura Municipal e a Universidade do Planalto Catarinense.

Referências bibliográficas

CÂMARA, G. **Modelos, Linguagens e Arquiteturas para Bancos de Dados Geográficos**. 264 f. 1995. Tese (Doutorado em Computação Aplicada) – Instituto Nacional de Pesquisa Espacial – INPE, São José dos Campos.

CARVALHO, M. S. (org.) **Sistemas de Informação Geográfica e a Gestão da Saúde no Município**. Brasília: Ministério da Saúde, 1999.

CONFEA. Muito além dos mapas. **Revista do Confea** - Conselho Federal de Engenharia, Arquitetura e Agronomia, ano VI, n. 10, mai./jun. 2002.

DPI/INPE. **Sistema de Processamento de Informações Georeferenciadas – SPRING**. São José dos Campos: DPI/INPE. Disponível em: <http://www.dpi.inpe.br/spring/>. Acessado em: 23 ago. 2005.

FEITOZA, H. N. **O uso de Geoprocessamento na Saúde Pública**. Relato de caso. Curitiba: MundoGEO, 2004.

HARA, L. T. 1997. **Técnicas de Apresentação de Dados em Geoprocessamento**. Dissertação (Mestrado em Sensoriamento Remoto) – INPE, São José dos Campos.

MAGALHÃES, M. *et al*. Experiência de ensino de geoprocessamento para profissionais de saúde, **I Simpósio Nacional de Geografia da Saúde**, Presidente Prudente, Brasil. 2003,

MARISCO,N.; PHILIPS, J.; PEREIRA, H. R. Protótipo de mapa para web interativo: uma abordagem utilizando código aberto. **Revista Brasileira de Cartografia**, n. 56/01, 2004.

PEDROSO, B. S. *et al*. Estudo da mortalidade infantil no Sul do RS com o uso do Geoprocessamento. **Revista do CCEI**, v. 6., n. 9. Bagé: URCAMP, 2002. p. 68-74.

PETERSON, M. P. **Interactive and Animated Cartography**. Nebraska/Omaha: Prentice-Hall, 1995.

UCHOA, H. N.; FERREIRA, P. R. **Geoprocessamento com Software Livre**. Disponível em: <http://www.geolivre.org.br/downloads/geoprocessamento_software_livre_Uchoa-Roberto - v1.0.pdf>. 17 out. 2004.

M3GE: um Motor de Jogos 3D para Dispositivos Móveis com Suporte a Mobile 3D Graphics API

Paulo César Rodacki Gomes (FURB/DSC)
rodacki@inf.furb.br

Vitor Fernando Pamplona (FURB/BCC)
vitor@babaxp.org

Resumo. Este artigo apresenta os primeiros resultados obtidos no processo de construção de um motor de jogos 3D para dispositivos móveis chamado Móbile 3D Game Engine (M3GE). O motor utiliza a especificação Mobile 3D Graphics API for J2ME (M3G) e possui recursos de tratamento de colisão, mapeamento de texturas, entre outros. É também apresentada a implementação de um protótipo simples de jogo 3D, utilizando o motor desenvolvido.

Palavras-chave: Motores de jogos, Dispositivos móveis, Celulares, M3G, OpenGL ES.

1 Introdução

Há muitos anos o homem cria jogos para se divertir. O jogo sempre foi sinônimo de competição, avaliando quem é o mais forte ou o mais rápido. A era tecnológica os evoluiu, criando ambientes complexos tanto para reprodução visual quanto para resolução do enredo. Atualmente, um novo setor está chamando a atenção, um segmento que está se desenvolvendo com muita rapidez, e tende a ultrapassar o volume de produção dos jogos atuais. Esta é a área de jogos para dispositivos móveis. Um mercado muito rico, amplo e diversificado (BATTAIOLA et al., 2001).

Os jogos para celulares disponíveis ao mercado podem ser comparados com os jogos existentes no final dos anos 80. Jogos simples, em duas dimensões e sem utilizar muitos recursos gráficos. Estimativas indicam que, em alguns anos, os jogos eletrônicos tridimensionais executados em micro-computadores estarão rodando em celulares, *palm tops*, *pocket pcs* e outros (AARNIO, 2004). Espera-se que esta evolução crie novos mercados, tais como o de desenvolvimento de arquiteturas e ferramentas para facilitar o desenvolvimento desses jogos. Um exemplo deste tipo de ferramenta são os motores de jogos. A plataforma *Java 2 Micro Edition* (J2ME) (SUN MICROSYSTEMS, 2004a) é uma versão simplificada das APIs do Java e da sua máquina virtual. A união entre dispositivos móveis e a tecnologia Java trouxe grandes resultados nas áreas de automação comercial e industrial, surgindo, no mercado, muitos sistemas com interfaces em celulares e PDAs. Porém, no desenvolvimento de jogos, o Java foi inicialmente descartado por ser muito lento. A adoção para este tipo de desenvolvimento foi maior em linguagens nativas dos dispositivos por serem mais rápidas e com mais recursos gráficos.

Apesar da tendência de crescimento do uso da plataforma, até o presente momento surgiram poucas iniciativas para o desenvolvimento de jogos 3D em dispositivos móveis em Java. Uma das mais recentes é a especificação *Mobile 3D Graphics API for J2ME* (M3G), proposta pela Nokia (NOKIA, 2003). Este artigo propõe o uso do J2ME e da M3G para implementação de motores de jogos 3D para dispositivos móveis. Os autores desconhecem propostas semelhantes na literatura e acreditam que esta ausência de referência se deve ao fato da especificação M3G ser ainda muito recente.

Na próxima seção é feita uma breve discussão sobre motores de jogos 3D. Os componentes da arquitetura proposta são apresentados na seção 3. A seção 4 apresenta a implementação de um

protótipo que visa ilustrar a viabilidade da presente proposta. Por fim, a seção 5 apresenta conclusões e trabalhos futuros. Como complemento deste artigo, a página web do projeto (`https://m3ge.dev.java.net`) disponibiliza os arquivos fonte.

2 Motores de jogos

Os motores são bibliotecas de desenvolvimento responsáveis pelo gerenciamento do jogo, das imagens, do processamento de entrada de dados e outras funções (EBERLY, 2001). A idéia é que os motores implementem funcionalidades e recursos comuns à maioria dos jogos de determinado tipo, permitindo que esses recursos sejam reutilizados a cada novo jogo criado (PESSOA, 2001). Os motores são tão importantes que estão em praticamente todos os jogos para micro-computadores, controlando a estrutura do jogo e seu ciclo de vida. A figura 1 exibe a arquitetura típica de um motor de jogos 3D (BATTAIOLA et al., 2001).
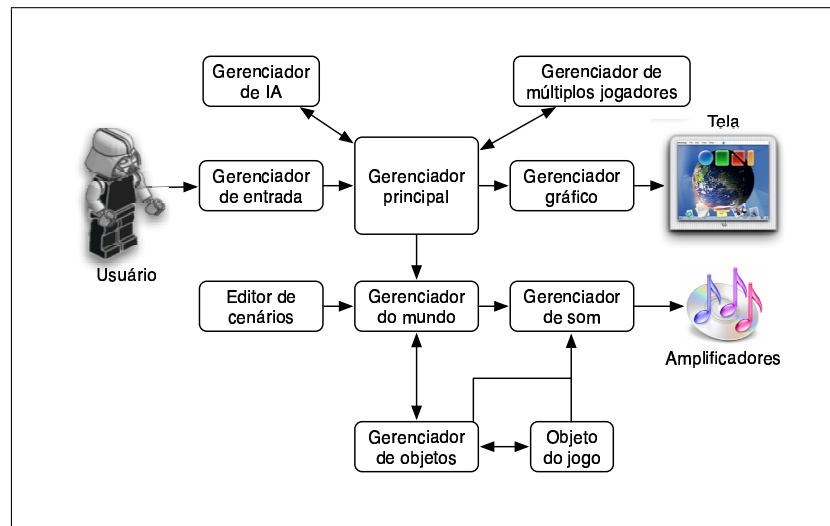
Figura 1: Exemplo de arquitetura de um motor de jogos

O gerenciador de entrada recebe e identifica os eventos de entrada e os envia para o gerenciador principal. O gerenciador gráfico transforma o modelo que define o estado atual do jogo em uma visualização para o usuário. O gerenciador de inteligência artificial gerencia o comportamento dos objetos desenvolvidos pelo *designer* do jogo. O gerenciador de múltiplos jogadores trata da comunicação dos jogadores, independentemente do meio físico em que se encontram. O gerenciador de objetos carrega, controla o ciclo de vida, salva e destrói um grupo de objetos do jogo. Em geral, um jogo possui vários gerenciadores de objetos que, além de suas funções normais, ainda precisam se comunicar. O objeto do jogo possui dados relevantes para uma entidade que faça parte do jogo (como ”avião”, ”monstro”, etc). Esta parte do motor controla a posição, velocidade, dimensão, detecção de colisão, entre outros. O gerenciador do mundo armazena o estado atual do jogo e para isso utiliza os gerenciadores de objetos. Em geral, uma ferramenta externa, o editor de cenários, descreve um estado inicial do jogo para cada um de seus níveis. O gerenciador principal é responsável pela coordenação entre os demais componentes.

Atualmente existem vários motores de jogos 3D disponíveis. A lista é extensa, e podemos citar alguns exemplos, tais como os projetos *open source Crystal Space* (CRYSTAL SPACE, 2004)

e *Ogre3D* (OGRE3D, 2005), além do *framework* proprietário *Fly3D* (PARALELO COMPUTAÇÃO, 2004). Para jogos em dispositivos móveis, os *frameworks* mais conhecidos para desenvolvimento utilizando linguagem nativa são o *ExEn* (RAMOS et al., 2003) e o *Mophun*, implementada em linguagem C (AMARO, 2005). Os motores que utilizam a máquina virtual Java são mais escassos, temos como exemplos o *wGem*, um dos primeiros motores de jogos para dispositivos móveis do Brasil (PESSOA, 2001).

Em 1998, com a criação do *Java Community Process* (JCP) (JCP, 2004a) a tecnologia Java deixa de ser propriedade da *Sun Microsystems* e passa a ser propriedade de um grupo de especificação, do qual qualquer empresa poderia pertencer. O JCP criou as *Java Specification Request* (JSR) (JCP, 2004b), especificações claras, concisas e livres, que determinam os padrões de desenvolvimento, novas implementações ou revisões de uma implementação existente no Java. Estes documentos permitem a outras empresas participarem ativamente do desenvolvimento da tecnologia Java, aumentando o foco tecnológico e abrindo espaço para que a tecnologia possa ser difundida.

Em 2002, o fabricante de telefones celulares Nokia lançou a JSR-184 (NOKIA, 2003) com o objetivo de criar rotinas gráficas velozes e práticas para substituir as implementações das linguagens nativas, a *Mobile 3D Graphics API for J2ME* (M3G). A diferença entre a maioria dos motores de jogos para dispositivos móveis citados acima e o motor proposto neste trabalho está no uso da M3G. A M3G ”define rotinas de baixo e alto nível para tornar eficiente e criar interatividade de gráficos 3D para dispositivos com pouca memória e poder de processamento, sem suporte de hardware ou para operações com pontos flutuantes” (MAHMOUD, 2004). Embora esta definição faça menção a dispositivos sem suporte de hardware para 3D, praticamente, apenas os dispositivos que implementam alguma função nativa da *Application Programming Interface* (API) 3D conseguem uma velocidade de renderização aceitável. Os celulares Nokia, por exemplo, utilizam uma implementação nativa do OpenGL ES (GROUP, 2004), uma versão simplificada do OpenGL. Outros fabricantes de dispositivos estão implementando estruturas similares. A M3G foi especificada para as versões *Mobile Information Device Profile* (MIDP) 2.0 e *Connected Limited Device Configuration* (CLDC) 1.1 (SUN MICROSYSTEMS, 2004a). O CLDC é uma configuração que define os recursos da máquina virtual e as bibliotecas principais para J2ME, e o MIDP consiste em um perfil para dispositivos portáteis definindo APIs como a de interface com o usuário, redes e conectividade, armazenamento, entre outros. A figura 2 ilustra a arquitetura básica da M3G:
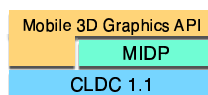
Figura 2: Arquitetura básica da M3G

A JSR 184 (M3G), definiu o seguinte conjunto de capacidades que a API deve suportar:

- funcionamento em *retained-mode*, importando os grafos de cena de algum lugar, ou em *immediate-mode*, permitindo ao desenvolvedor criar seus próprios grafos de cena;
- a API deve importar malhas de polígonos 3D, texturas e grafos de cena;
- os dados devem estar em formato binário para diminuir o tamanho do armazenamento e a transmissão;
- deve ser possível implementar a API sobre a OpenGL ES, sem recursos de ponto flutuante de hardware;

- a API deve implementar valores com ponto flutuante;
- memórias ROM e RAM ocupadas devem ser mínimas. A API deve ocupar menos de 150 KB;
- a API deve implementar algum mecanismo de *garbage collection*;
- a API deve ser inter-operável com outras APIs Java, especialmente o MIDP.

A API está definida para ser implementada dentro do pacote *javax.microedition.m3g* contendo 30 classes divididas em 6 grupos: classes básicas, classes para nós de grafo de cena, classes para carga de arquivos e funcionalidades de baixo nível, classes para atributos visuais, classes modificadoras, e, por fim, classes para animação e tratamento de colisão. A seguir é apresentada a proposta do motor de jogos utilizando a M3G.

3 O motor de jogos M3GE

O motor de jogos proposto no presente artigo é chamado *Mobile 3D Game Engine* (M3GE) e foi desenvolvido seguindo as três etapas básicas de construção de software: análise, implementação e testes. Na fase de análise, foram levantados os seguintes requisitos principais:

- carregar e desenhar um ambiente 3D a partir de um arquivo de configuração;
- criação de um número indefinido de câmeras, que podem ser trocadas dinamicamente durante o jogo;
- tratamento de eventos do usuário;
- movimentação de personagens no cenário, com seis graus de liberdade;
- portabilidade, oferecida pela linguagem Java;
- desempenho para renderização em tempo real, com mapeamento de texturas;
- tratamento de colisão.

A figura 3 ilustra a arquitetura proposta pelos autores. Como pode ser visto, a M3GE foi projetada como uma API anexa à M3G. Ou seja, mesmo usando a M3GE, é possível utilizar a M3G diretamente. As duas bibliotecas interagem entre si, proporcionando ao desenvolvedor do jogo uma maior flexibilidade e velocidade, pois pode-se acessar o OpenGL ES diretamente quando necessário. Esta opção de projeto procura atender o requisito de desempenho para renderização 3D citado anteriormente.
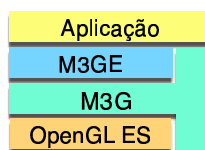
Figura 3: Arquitetura básica da M3G

Em comparação com a arquitetura típica de motores de jogos 3D apresentada na figura 1, a M3GE implementa elementos escurecidos da figura 4.
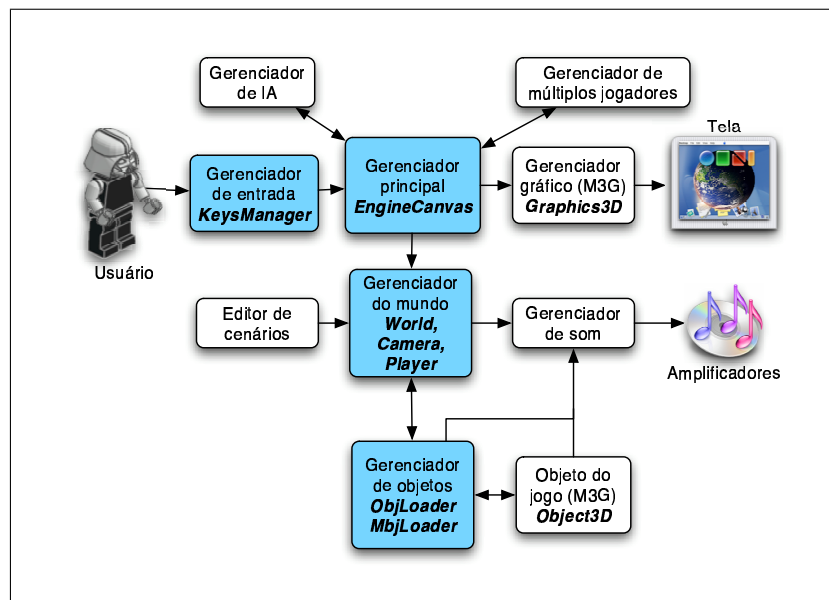
Figura 4: Módulos implementados pela M3GE

O projeto de implementação da M3GE foi dividido em dois grandes componentes: o responsável pela leitura dos arquivos de entrada e o responsável pelo motor de jogos propriamente dito, ou *core*, como mostrado na figura 5.

A construção do cenário do jogo é feita pela leitura de um arquivo de configurações gerais do ambiente, um arquivo com as malhas de polígonos e uma série de arquivos de texturas Conforme será visto na seção 3.2. A seguir, são detalhadas as classes do motor de jogos proposto.

3.1 Classes do *core*

A classe *KeysManager* atua como gerenciador de entrada, recebendo eventos do usuário quando ele digita alguma tecla do dispositivo. Esta classe verifica se existem métodos atribuídos a cada tecla pressionada e, quando existir, chama os objetos responsáveis pela ação de acordo com cada tecla. Para que isto aconteça, a informação referente a um evento passa para a classe *EngineCanvas*, que é o gerenciador principal do motor de jogos 3D. Ela é responsável pelo gerenciamento do ciclo de vida de todos os objetos, pela chamada do *KeysManager* a cada tecla pressionada, pela renderização da cena utilizando a M3G, pela carga dos arquivos de configuração, acionada pelo construtor da classe, e pela criação de câmeras. Esta classe trabalha com instâncias das classes *Player*, *Cameras* e *World*, que foram construídas separando as responsabilidades de um único gerenciador de mundo, mas atuando em conjunto.

A classe *Player* implementa o personagem principal do jogo, controlado pelo usuário. Esta classe mantém posição, ângulos e tamanho, assim como a geometria do personagem. O *Player* é um grupo de nós do grafo de cena da M3G, e, com isso, pode manter o desenho do personagem em seus nós filhos. Quando necessário, o *Player* também é responsável por chamar as rotinas de teste de colisão. O método *update(KeysManager keys)* movimenta o personagem de acordo com as teclas pressionadas, testando colisão entre o personagem e os demais objetos e a colisão de tiro com algum objeto no modelo.
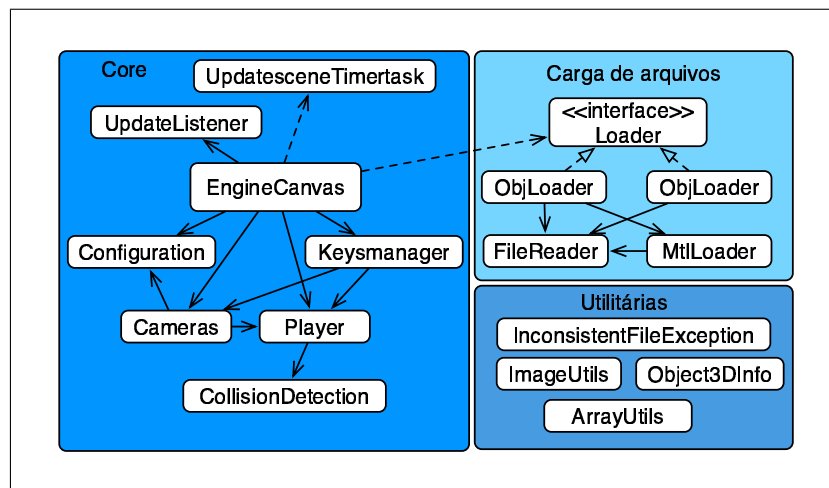
Figura 5: As classes da M3GE

A classe *UpdateListener* deve ser implementada pelo desenvolvedor do jogo para o caso de haver necessidade de tratar alguma ação nas rotinas de renderização, ou alguma lógica de jogo a ser adicionada em termos de eventos. Isto permite, por exemplo, que o desenvolvedor de jogos detalhe informações para os jogadores, escrevendo-as diretamente na tela. Os principais eventos disponíveis são o evento camUpdate, gerado quando o jogador aciona a troca de câmera, o evento fire gerado quando o jogador atira e acerta algum objeto 3D, *keyPressed* e *keyReleased*, gerados quando o usuário pressiona e solta alguma tecla do dispositivo, *update*, gerado antes de ser feito o redesenho da cena e *paint* gerado após o redesenho da cena. Este evento permite, por exemplo, que seja desenhado algum elemento 2D no dispositivo gráfico, após o desenho da cena 3D.

O gerenciador gráfico já é implementado pela M3G, e é representado pela classe *Graphics3D*. Sua responsabilidade é desenhar o modelo 3D em um dispositivo gráfico 2D. As classes *Configuration* e *CollisionDetection* são responsáveis por carregar os arquivos de configuração do motor e fazer o tratamento de colisões. A detecção de colisão, efetuada pela classe *CollisionDetection*, foi implementada na forma mais simples possível, procurando não perturbar a velocidade da renderização das cenas e não prejudicar a jogabilidade. São feitos testes de colisão em três pontos, um no centro e a frente do jogador e os outros dois nas laterias conforme ilustrado no modelo de detecção de colisão ilustrado na figura 6, onde R representa o parâmetro *CollisionRay* é informado no arquivo de configurações do ambiente do jogo. Esta classe também implementa o cálculo de colisão de tiro disparado pelo personagem contra objetos 3D do ambiente.

A classe *Cameras* é responsável por manter toda a estrutura de câmeras do jogo, carregando as configurações de arquivo, gerenciando o posicionamento de cada câmera no mundo e identificando qual a câmera atualmente utilizada para renderizar as imagens. A classe mantém um vetor com os parâmetros de cada uma das câmeras do jogo, e um atributo listener que mantém uma referência para um objeto de uma classe implementada pelo desenvolvedor do jogo a ser invocada assim que ocorrer uma troca de câmeras. Uma câmera é definida por seu ponto de localização (x, y, z), pelas coordenadas de seu target (ax, ay, az), pelo ângulo de visão $fovy$ e pelos planos de corte do *frustrum* de visualização far e $near$. A classe *Object3DInfo* armazena as coordenadas do ponto central de cada objeto tridimensional e o seu nome. Além disso, esta categoria de classes contém classes utilitárias para gerenciamento dos *arrays*, imagens e tratamento de excessões na leitura dos
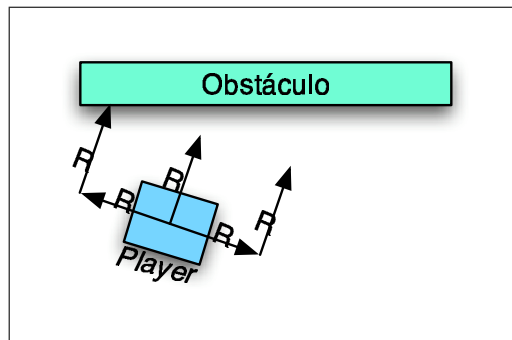
Figura 6: Modelo de detecção de colisão

arquivos. Por fim, a classe *UpdateSceneTimerTask* implemeta um mecanismo de chamada às rotinas de desenho do motor em um determinado intervalo de tempo.

3.2 Classes de carga de arquivo

As classes *ObjLoader* e *MbjLoader* atuam como gerenciadores de objetos, criando instâncias da classe *Object3D* e dos nós de grafo de cena, e gerenciando o seu ciclo de vida. A M3G especifica o formato *Wavefront* como padrão para arquivos de entrada de dados para criação do cenário e configuração do ambiente (O'REILLY & ASSOCIATES INC, 1996). Este formato foi criado para ser utilizado com o *Wavefront Advanced Visualizer* da Viewpoint DataLabs. O modelo 3D é separado em dois arquivos ASCII. O arquivo *obj* descreve a geometria, podendo conter uma série de tipos de primitivas gráficas. No presente trabalho, apenas as malhas de polígonos 3D são consideradas e lidas pela M3GE. Este arquivo também declara grupos, que são coleções de polígonos 3D formando objetos 3D na cena. O segundo arquivo, de extensão *mtl*, descreve cores, propriedades de materiais e texturas do ambiente ou de um grupo de objetos 3D. As texturas são armazenadas em arquivos de imagem separados. De acordo com a especificação da M3G as imagens devem ser quadradas, com largura e altura medindo potências de 2 pixels. Para que a importação de um arquivo *obj* seja viável considerando as limitações de memória e processamento dos telefones celulares atuais, os autores propõem que a sua definição seja extendida adicionando algumas características: os vértices devem ser obrigatoriamente 3D e as faces devem ser triangulares, todos os dados de textura e cores devem ser colocados num único arquivo *mtl*, deve haver a mesma quantidade de vértices, vetores normais, e vetores de textura e eles devem ser igualmente seqüenciados no arquivo.

A carga completa do grafo de cena é feita em três leituras do arquivo *obj* pela classe *ObjLoader*. A leitura inicial conta a quantidade de vértices, vetores normais e texturas para criar o *array* de vetores para segunda leitura. Foram cogitadas algumas outras possibilidades, como a utilização de estruturas de dados como a classe Vector do J2ME, mas esta estrutura recria todo o seu array interno a cada 16 posições o que torna inviável sua utilização na M3GE. O uso de outras estruturas, tais como listas encadeadas também é inviável, devido ao *overhead* de memória necessário. Na segunda leitura do arquivo são obtidas as coordenadas dos vértices, dos vetores normais e dos vetores de textura, além de referências aos dados dos arquivos *mtl* para cada grupo (objeto 3D) da cena, que são importados em uma única leitura do arquivo *mtl*. As coordenadas no arquivo estão em tipos de dados *float* (número com ponto flutuante e precisão simples) e são convertidas internamente para tipo *byte*. Após isso, é feita uma terceira leitura do arquivo *obj* para a criação dos objetos 3D no jogo, a partir dos grupos no arquivo. Neste momento, é calculado o ponto central de cada objeto 3D.

O arquivo *Wavefront* não foi especificado para ser utilizado por dispositivos móveis e, por

```
nf <int>
nv <int>
nvt <int>
nvn <int>
mtlib <arquivo.mtl>
v vt vn
<byte> <byte><byte><byte><byte><byte><byte><byte>
g <float> <float> <float> <nome>
usemtl <nome de um material>
f <int[/int[/int]]>[<int[/int[/int]]>[<int[/int[/int]]>]] ...
```

Quadro 1: O formato de arquivos *mbj*

conseqüência, este processo de leituras repetidas acaba acarretando demora na montagem do cenário do jogo. Em testes feitos pelos autores, a carga de um arquivo *obj* relativamente pequeno, com 2000 linhas, em um telefone celular Siemens CX65 demorou em média 17 segundos até sua apresentação na tela do aparelho. Os celulares Siemens possuem otimizações para re-leituras de arquivos tais como *cache* de arquivos, porém dispositivos de outros fabricantes podem não se comportar da mesma maneira, o que aumentaria significativamente o tempo de carga. Para minimizar este problema, o presente trabalho propõe a especificação de um arquivo semelhante ao *Wavefront*, mas com algumas limitações. Nesta especificação, os dados devem ser do tipo byte ao invés do tipo float com as coordenadas dos pontos centrais pré-calculadas e com informações sobre o tamanho dos arrays a serem montados. O arquivo recebeu a extensão *mbj*, que significa, *Mobile Object File* e sua estrutura é mostrada no quadro 1.

Os atributos nf, nv, nvt e nvn indicam respectivamente as quantidades de faces, vértices, vetores de textura e vetores normais. A seguir, são listadas as coordenadas x, y e z de cada vértice, as coordenadas u e v das texturas e as coordenadas dos vetores normais. Por fim, vêm as declarações de grupos, materiais e faces.

Para ler um arquivo *mbj* dentro do dispositivo, é utilizada a classe *MbjLoader*, muito semelhante a *ObjLoader*, mas sem fazer nenhum cálculo ou conversão de tipos de dados encontrados no arquivo. A leitura é mais rápida que a do formato anterior, principalmente, nos celulares que não implementam nativamente algum tipo de cache de arquivos, evitando o acesso a memória ROM a partir da primeira leitura. Para carregar um modelo pelo arquivo *Wavefront obj* são necessárias três leituras, o que pode triplicar o tempo de carga do jogo se o celular não implementar este cache. Mesmo com o cache de arquivos, a diferença entre a carga do mesmo modelo de testes nos dois formatos de arquivo ficou em torno de 2 segundos.

4 Implementação de um jogo simples

Neste trabalho um protótipo de jogo simples é implementado para testar e demonstrar o funcionamento do motor M3GE desenvolvido. Ao contrário das outras aplicações, um jogo é feito com duas unidades de processamento distintas, como pode ser visto na figura 7. Uma delas atua sobre a camada de modelo do padrão de projeto *Model-View-Controller* (MVC) (SUN MICROSYSTEMS, 2004c) e é enquanto a outra atua sobre as camadas de controle e visão.

Para carregar um cenário e permitir que um personagem ande sobre ele, deve-se criar uma instância da classe *EngineCanvas* passando referências para os arquivos de entrada de dados. O
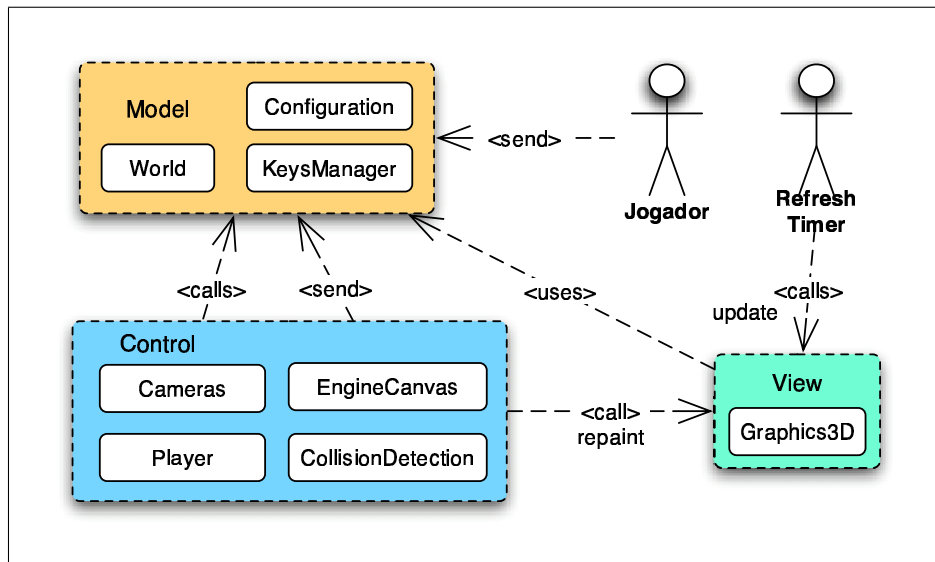
Figura 7: Um jogo com M3GE em MVC

protótipo implementado possui 7 objetos 3D (incluindo um personagem), 4 câmeras e 6 texturas de 256x256 *pixels* cada. Foi utilizado tratamento de colisão do personagem e implementado o disparo de tiros. A figura 8 exibe a visualização do protótipo num emulador de aparelhos celulares. Nas telas da esquerda para a direita tem-se a visão em 1^a pessoa do personagem, a visão em 3^a pessoa, uma visão de topo e uma visão genérica.

Figura 8: Visualização das diferentes câmeras do protótipo

Tanto a M3GE quanto o protótipo foram implementados no Eclipse com o plugin EclipseME (ECLIPSEME TEAM, 2005) e *Sun Wireless Toolkit* (SUN MICROSYSTEMS, 2004b). Os testes foram realizados com emuladores do *Siemens Wireles Toolkit* (SIEMENS AG, 2005) e a implementação de referência da M3G da Nokia que já vem com emuladores próprios (NOKIA, 2003). Além dos emuladores, foram feitos testes no aparelho celular Siemens CX65. Alguns emuladores limitavam a memória utilizada, enquanto que os aparelhos reais não. No quesito velocidade, a leitura de um modelo 3D leva alguns segundos de diferença entre um emulador e um celular, e varia de aparelho

para aparelho. A velocidade de jogo foi praticamente igual nas duas plataformas, exceto o tempo de compilação na primeira execução dos *bytecodes* Java. No telefone celular Siemens CX65 são necessários três segundos após o jogador tentar mover o personagem pela primeira vez. Este é o tempo em que a máquina virtual do celular otimiza os códigos deixando-os em cache para facilitar no ciclo de atualização das cenas na tela. Além disso, o aparelho gastou aproximadamente 6 segundos para ler e descompactar as texturas em formato *jpeg*. A carga de arquivos *obj* demorou 17 segundos, enquanto que o arquivo *mbj* foi carregado em 15 segundos. Considerando a totalidade dos arquivos, estudos mais detalhados identificaram que os celulares Siemens demoram de 100 a 900 milesegundos para localizar e abrir um arquivo dependendo de seu tamanho, assim concluímos que a abertura dos arquivos pela máquina virtual Java consome grande parte do tempo de processamento. A taxa de renderização ficou em torno de 8 *frames* por segundo para movimentação do personagem e 17 *frames* por segundo para a rotação do personagem, onde não existe teste de colisão.

Um primeiro jogo comercial utilizando a M3GE está sendo desenvolvido pela empresa norte americana Autonomous Productions (GUILIANO, 2005). O jogo, cujo provável nome comercial é Tranquility, deverá ser lançado no mercado nos próximos meses, e consiste em um simulador de vôos 3D. A figura 9 apresenta uma tela do jogo no emulador do WTK (SUN MICROSYSTEMS, 2004b).

Figura 9: Simulador de vôo em desenvolvimento

5 Conclusões

O presente trabalho apresentou a especificação e o desenvolvimento de um motor de jogos 3D em java para celulares com suporte à especificação *Mobile 3D Graphics API for J2ME*. O trabalho também definiu e implementou um formato de arquivos especial para facilitar a importação de modelos 3D e uma aplicação para a conversão de arquivos *Wavefront* para o formato *mbj*. A partir de resultados obtidos em testes realizados em dispositivos reais, pode-se concluir que a tecnologia Java pode ser utilizada para construir jogos 3D em dispositivos limitados, embora a preocupação com algoritmos velozes seja sempre necessária pois construir aplicações com poder de processamento e memória muito limitados é muito diferente de desenvolver aplicações normais, para microcomputadores.

A viabilidade da proposta é apresentada através da modelagem e implementação de um protótipo de jogo simples. Não se trata de um trabalho completo, mas sim de uma primeira experiência, visto que a M3GE ainda não implementa uma série de funcionalidades desejáveis tais como força gravitacional, dinâmica de corpos rígidos e outras. Mesmo assim, um primeiro jogo comercial já está sendo desenvolvido com a primeira versão da M3GE, disponibilizada sob licença GPL. Como proposta de continuação da presente pesquisa, sugere-se o desenvolvimento de um *framework* com uma biblioteca de classes mais completa. Ainda como sugestão de trabalhos futuros, aponta-se a necessidade de testes de desempenho com diferentes modelos de aparelhos celulares.

6 Agradecimentos

Os autores agradecem ao Sr. Marcelo Eduardo M. de Oliveira, do Instituto Nokia de Tecnologia e ao Dr. Andrew Davison pela troca de informações e pelo material de consulta disponibilizado.

Referências

AARNIO, Tomi. **A new dimension for Java games**: mobile 3d graphics api. [P.O.Box], 2004. Disponível em: <http://www.nokia.com/nokia/0,,62395,00.html>. Acesso em: 12 set. 2004.

AMARO, Pedro H. S. **The clash of mobile platforms**: J2me, exen, mophun and wge. [Coimbra, Portugal], 2005. Disponível em: <http://www.gamedev.net/reference/articles% -/article1944.asp>. Acesso em: 15 ago. 2005.

BATTAIOLA, André L. et al. Desenvolvimento de jogos em computadores e celulares. **Revista de Informática Teórica e Aplicada**, v. 8, n. 2, out. 2001.

CRYSTAL SPACE. **Crystal Space 3D**. [S.l.], 2004. Disponível em: <http://crystal.sourceforge-.net>. Acesso em: 30 set. 2004.

EBERLY, David H. **3D game engine design**: a practical approach to real-time computer graphics. São Francisco: Morgan Kaufmann, 2001.

ECLIPSEME TEAM. **EclipseME home page**. [Scottsdale], 2005. Disponível em: <http:/-/eclipseme.org/index.html>. Acesso em: 25 maio 2005.

GROUP, KHONOS. **OpenGL ES**: overview. [San Francisco], 2004. Disponível em: <http://www.opengles.org/opengles/index.html>. Acesso em: 04 ago. 2005.

GUILIANO, Shayne. **Autonomous productions**. [S.l.], 2005. Disponível em: <http://www-.autonomousproductions.com/website/index.html>. Acesso em: 14 set. 2005.

JCP. **The Java community process(SM) program**: JCP procedures - JCP 2 process document. [Palo Alto], 2004a. Disponível em: <http://www.jcp.org/en/procedures/jcp2>. Acesso em: 30 set. 2004.

JCP. **The Java community process(SM) program**: JSRs - Java specification requests - JSR overview. [Palo Alto], 2004b. Disponível em: <http://www.jcp.org/en/jsr/overview>. Acesso em: 28 out. 2004.

MAHMOUD, Qusay H. **Getting Started with the Sobile 3D Graphics API for J2ME**. [Palo Alto], 2004. Disponível em: <http://developers.sun.com/techtopics/mobility/apis/articles/3dgraphics/>. Acesso em: 30 out. 2004.

NOKIA. **JSR-184 mobile 3D API for J2ME**. [P.O.Box], 2003. Disponível em: <http://www-.forum.nokia.com/main/0,6566,040,00.html>. Acesso em: 11 set. 2004.

OGRE3D. **OGRE 3D**: open source graphics engine. [S.l.], 2005. Disponível em: <http://www-.ogre3d.org>. Acesso em: 15 jul. 2005.

O'REILLY & ASSOCIATES INC. **GFF format summary**: wavefront obj. [S.l.], 1996. Disponível em: <http://netghost.narod.ru/gff/graphics/summary/waveobj.htm>. Acesso em: 10 maio 2005.

PARALELO COMPUTAÇÃO. **Fly3D.com.br**. [Niterói], 2004. Disponível em: <http://www-.fly3d.com.br>. Acesso em: 2 out. 2004.

PESSOA, Carlos A. C. **wGEM**: um framework de desenvolvimento de jogos para dispositivos móveis. 2001. Dissertação (Mestrado) — UFPE.

RAMOS, Otavio R. et al. A mobile device game development initiative in academia: Challenges and preliminary results. In: **Proceedings of WJogos 2003**. Porto Alegre: SBC, 2003.

SIEMENS AG. **Siemens communications group**. Munich, 2005. Disponível em: <https://communication-market.siemens.de/portal/main-.aspx?LangID=0MainMenuID=10ParentID=10LeftID=30pid=1cid=0tid= 3000xid=0>. Acesso em: 25 maio 2005.

SUN MICROSYSTEMS. **Java 2 platform, micro edition (J2ME)**: Jsr 68 overview. [Palo Alto], 2004a. Disponível em: <http://java.sun.com/j2me/overview.html>. Acesso em: 10 set. 2004.

SUN MICROSYSTEMS. **Java 2 platform micro edition, wireless toolkit**. [Palo Alto], 2004b. Disponível em: <http://java.sun.com/products/j2mewtoolkit% -/index.html>. Acesso em: 19 set. 2004.

SUN MICROSYSTEMS. **Java BluePrints**: model-view-controller. [Palo Alto], 2004c. Disponível em: <http://java.sun.com/blueprints/patterns/MVC-detailed.html>. Acesso em: 20 set. 2004.

Animação de um Personagem Virtual Utilizando Captura Óptica de Movimento com Marcações Especiais

Giovane Roslindo Kuhn (FURB)
brain@netuno.com.br

Paulo César Rodacki Gomes (FURB)
rodacki@inf.furb.br

Resumo. Este trabalho utiliza os conceitos para a captura óptica de movimento humano (MoCap), utilizando marcações especiais sobre o corpo do ator, para a animação de um personagem virtual 3D. É apresentando o desenvolvimento de um protótipo de software que implementa as etapas de um sistema MoCap, gerando arquivos no formato BVH contendo a animação. Para isso, são aplicadas técnicas de processamento de imagens, no intuito de segmentar os marcadores, em seguida são aplicadas técnicas de predição para rastrear os marcadores ao longo de vídeo. Após isto, um modelo articulado é encaixado aos marcadores, para então ser rastreado ao longo vídeo e gerar o arquivo BVH com a animação do personagem virtual. Para o rastreamento do modelo é proposto um algoritmo que utiliza relações matemáticas entre as articulações do modelo para efetuar os seus posicionamentos.

Palavras-chave: Animação, BVH, Captura de movimento, Estruturas articuladas, MoCap, Personagem virtual

1 Introdução

O conceito de animação é antigo e analisando as suas origens, nota-se que os primeiros trabalhos nesta área são datados muito antes de invenção dos computadores. Segundo Parent (2002), a definição mais simples para animação é a geração de uma seqüência de imagens que retrata o movimento relativo de objetos de uma cena sintética, e possivelmente, do movimento de uma câmera virtual.

O início das experiências com animação por computadores, representou um grande avanço nas técnicas de animação. Uma das técnicas mais utilizadas na atualidade é a animação através de captura de movimento (SILVA, 1998), também chamada de *Motion Capture* (MoCap). O foco de estudo deste trabalho é especificamente a captura óptica de movimento, que consiste em colocar marcadores especiais no objeto durante o processo de captura do seu movimento e capturar as coordenadas destes marcadores através do uso de técnicas de processamento de imagens.

A alta qualidade do movimento gerado pela captura óptica de movimento, desperta interesse nas mais diversas áreas, como a indústria cinematográfica, jogos, ergonomia, desempenho desportivo e análise de movimento (PARENT, 2002). Porém utilizar um sistema MoCap ainda é uma realidade distante da grande maioria das empresas, universidades e grupos de pesquisa em nosso país, devido o alto custo dos *softwares* e *hardwares* envolvidos.

O grupo de pesquisa de ergonomia da Universidade Regional de Blumenau (FURB) desenvolve projetos utilizando análise de postura e movimento de pessoas, que necessitam ser enviados para a Universidade do Sul de Santa Catarina (UNISUL), localizada em Florianópolis, para serem analisados pelo sistema existente naquela instituição. O presente trabalho pretende contribuir neste sentido, dando continuidade as pesquisas em captura de movimento humano no Departamento de Sistemas e Computação.

Para atender esta necessidade, é implementado um protótipo de *software* para animar um personagem virtual utilizando a captura óptica de movimento. O protótipo utiliza o conjunto de

vídeos disponibilizados pelo grupo de pesquisa de ergonomia da FURB. Cada vídeo contém a gravação em plano sagital de uma criança caminhando, com marcadores especiais nas principais articulações do seu corpo. O plano sagital corresponde a uma visão lateral da pessoa a ser filmada.

A Seção 2 apresenta conceitos de animação e captura de movimento utilizados no trabalho, e um detalhamento do formato de arquivo BVH. A seguir, a Seção 3 apresenta aspectos sobre desenvolvimento do protótipo, detalhando as etapas especificadas e implementadas do sistema MoCap. Por fim, na Sessão 4 são apresentadas as conclusões deste trabalho.

2 Captura de movimento e animação

Esta seção tem por objetivo apresentar os conceitos de animação e captura de movimento utilizados neste trabalho, através de modelos para animar personagens, as etapas no processo de captura de movimento e o formato de arquivo BHV, utillizado para armazenar dados da animação e captura de movimento.

2.1 Personagem virtual

Em sistemas de animação, uma das primeiras etapas do ciclo de desenvolvimento é definir o modelo do personagem virtual, isto é, definir como o ator a ser animado é representado no computador e que parâmetros serão informados para animar este personagem (SILVA, 1998).

Geralmente o personagem virtual é modelado com estruturas articuladas, que consiste em um conjunto de objetos rígidos conectados por articulações. Estas articulações formam o vínculo geométrico entre os objetos, permitindo o movimento relativo entre eles.

No computador estruturas articuladas são representadas por estruturas hierárquicas (árvores), onde a posição de cada articulação é definida através da composição em seqüência das transformações das articulações anteriores. Com isto, apenas a primeira articulação da estrutura precisar ser posicionada no espaço, enquanto o resto da estrutura é posicionado apenas pelos ângulos entre as articulações, chamados de ângulos relativos. São estas informações, posição da articulação e ângulo relativo entre o restante das articulações, que são os parâmetros da animação do personagem virtual .
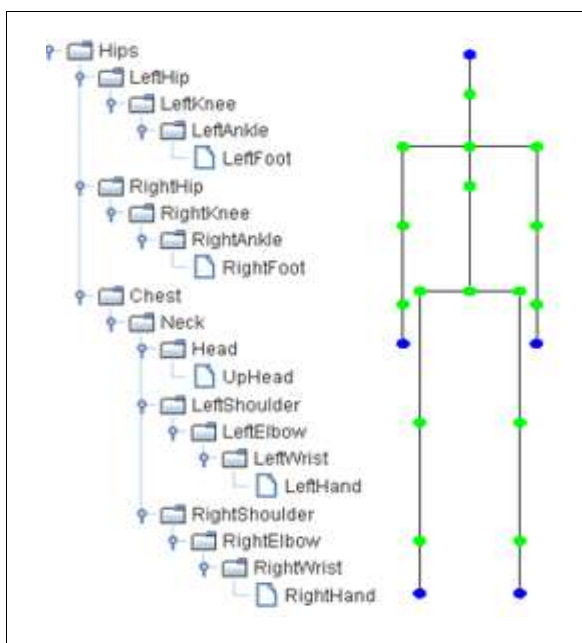
Figura 1: Topologia do personagem virtual

A fig. 1 apresenta a topologia do personagem virtual utilizado neste trabalho, que possui 16 articulações e 20 segmentros rígidos. A raíz da estrutura é o quadril do personagem e qualquer transformação nesta articulação tem efeito sobre toda a estrutura hierárquica.

2.2 Captura de movimento

A primeira etapa na captura de movimento, consiste em vestir o ator com marcadores reflexivos nos locais do corpo que se deseja capturar e posteriormente gravar o movimento do

ator, utilizando preferencialmente câmeras digitais de alta resolução e taxa de amostragem. Esta etapa não é realizada pelo presente trabalho, pois a captura do movimento do ator foi efetuada pelo grupo de pesquisas de ergonomia da FURB.

A etapa seguinte é efetuar processamentos nos dados capturados pelas câmeras para extrair o posicionamento 3D dos marcadores, para isto, técnicas de processamento de imagens são aplicadas em cada quadro do vídeo para realçar e segmentar os marcadores. Depois de segmentados os marcadores, são utilizadas técnicas de triangulação para efetuar o posicionamento 3D dos marcadores, com base nas informações 2D de cada câmera. O presente trabalho não precisou utilizar técnicas de triangulação, pois os vídeos disponibilizados gravam em plano sagital o movimento do ator.

A ultima etapa consite em gerar a trajetória dos marcadores entre os quadros do vídeo e encaixar o modelo do personagem virtual a estes marcadores rastreados. Os *softwares* de mercado não mencionam as técnicas computacionais utilizadas nesta etapa, o que se menciona, é que quando marcadores chaves não são rastreados, devido uma oclusão do marcador, por exemplo, é necessário que o usuário auxilie o *software* no rastreamento do marcador.

2.3 Padrão BVH

O BVH é um formato de arquivo para armazenar dados da captura de movimento e/ou animação. A escolha deste tipo de arquivo para o presente trabalho, deve-se ao fato do formato ser um padrão entre *softwares* de captura de movimento e suportado pelas principais ferramentas de animação do mercado, como SoftImage e 3D Studio MAX (BAERLE e THINGVOLD, [2005?]).

Um arquivo BVH é dividido em duas partes principais: dados da estrutura hierárquica do modelo e dados da animação deste modelo. O quadro 1 apresenta um exemplo de arquivo BVH, onde é possível notar a distinção entre os dados do modelo e da animação. Este exemplo está baseado na topologia do personagem virtual utilizado neste trabalho.

```
HIERARCHY
ROOT Hips
{
  OFFSET 0 0 0
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT LeftHip
  {
    OFFSET 15 0 -0
    CHANNELS 3 Zrotation Xrotation Yrotation
    ...
        End Site
        {
          OFFSET 0 -20 0
          CHANNELS 0
        }
    ...
}
MOTION
Frames: 95
Frame Time: 0.0400
0.0 195.0 35.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.0 0.0 0.0 0.0 0.0
...
```

Quadro 1: Exemplo de arquivo BVH

A palavra `HIERARCHY` define o início da estrutura hierárquica, `ROOT` define a articulação raiz desta estrutura, `End Site` indica os end-effectors da estrutura e `JOINT` o restante das articulações. O `OFFSET` indica a posição relativa (X, Y e Z) da articulação em relação ao seu pai, `CHANNELS` indica os parâmetros que serão animados da articulação, o BVH suporta seis tipos de parâmetros:

- `Xposition`, `Yposition` e `Zposition`: indicam a posição relativa desta coordenada em relação ao seu pai na hierarquia;

- `Xrotation`, `Yrotation` e `Zrotation`: indicam o ângulo de rotação sobre o respectivo eixo, a ordem em que aparecem na definição deve ser respeitada no momento da animação.

A palavra `MOTION` indica o início dos dados da animação, `Frames` a quantidade de quadros da animação, `Frame Time` o tempo de duração de um quadro em segundos e depois cada linha contém os valores dos parâmetros de um quadro da animação.

Os parâmetros apresentados neste exemplo são aqueles comumente utilizados em arquivos BVH, onde apenas a raiz da estrutura contém dados translacionais e o restante das articulações contém apenas dados rotacionais. Porém, o formato permite definir parâmetros translacionais para todas as articulações, um problema do formato translacional é que a distância entre as articulações pode variar durante a animação.

3 Desenvolvimento do protótipo

O projeto recebe o nome de *Apollo* (APOLLO, 2005) e utiliza a metodologia de desenvolvimento em espiral, na qual especificação e implementação são construídas em conjunto. Para especificação é utilizada a notação *Unified Modeling Language* (UML) em conjunto com a ferramenta *Java UML Modeling Tool* (JUDE, 2005) e conceitos de análise orientada a objetos.

Na implementação do protótipo de *software* é utilizada a linguagem de programação *Java* (JAVA, 2005) e o ambiente de desenvolvimento *Eclipse* (ECLIPSE, 2004). Para trabalhar com vetores é utilizada a biblioteca *Vecmath* do *Java3D* (JAVA3D, 2003).

Na manipulação dos vídeos é utilizada a biblioteca *Java Media Framework* (JMF, 2003). Os visualizadores utilizam a biblioteca *Java bindings for OpenGL* (JOGL, 2005) que segue a especificação *OpenGL 2.0* (Akeley e Segal, 2004).

3.1 Diagrama de atividades

O usuário do protótipo exerce dez atividades básicas para poder capturar o movimento de um ator em um vídeo e então animar um personagem virtual 3D. A fig. 2 apresenta o diagrama com as principais atividades efetuadas durante o processo.
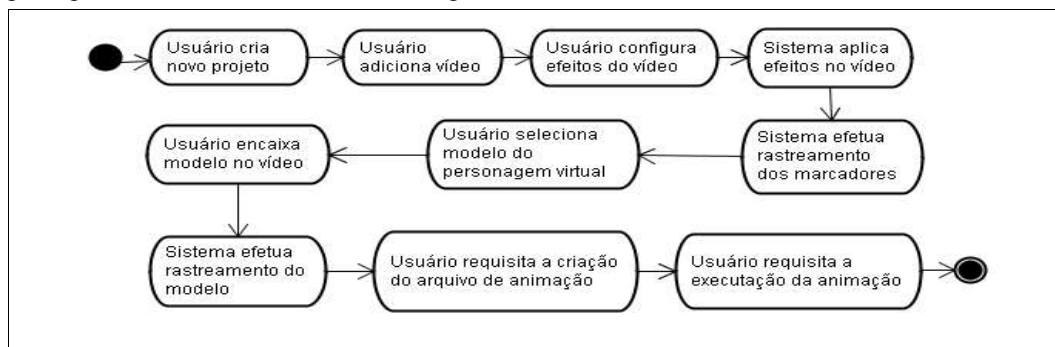
Figura 2: Diagrama de atividades principal

O fluxo de atividades principal consiste no usuário criar um novo projeto, neste projeto o usuário adiciona vídeos a serem analisados durante o processo de captura de movimento. O usuário tem condições de configurar os efeitos a serem aplicados no vídeo, no intuito de realçar os marcadores no corpo do ator, com isto, facilitar o segmentação destes marcadores.

Após configurado os efeitos, o usuário requisita para o sistema aplicar estes efeitos no vídeo, é neste momento que os marcadores são segmentados, isto é, a coordenada de cada marcadores é

extraída em cada quadro do vídeo. Com isto, o sistema pode efetuar o rastreamento destes marcadores, que consiste em analisar a trajetória dos marcadores ao longo do vídeo.

Feito o rastremanto dos marcadores, o usuário seleciona o modelo do personagem virtual a ser utilizado na animação e enaixa este modelo nos marcadores segmentados na etapa anterior. Após encaixar o modelo, o usuário requisita para rastrear este modelo ao longo do vídeo, enfim, definir a posição de cada articuação para cada quadro do vídeo.

Depois de rastrado o modelo, o usuário requisita a criação do arquivo BVH com a animação do personagem virtual e por fim requisita ao sistema para executar e então poder visualizar a animação do personagem virtual 3D.

3.2 Módulos

Com base nos requisitos levantados e no fluxo de atividades do projeto, o protótipo é dividido em cinco módulos: núcleo do protótipo, módulo de processamento do vídeo, módulo de rastreamento dos marcadores, módulo de rastreamento do modelo e módulo de animação.

No núcleo do protótipo é mantida a tela principal do aplicativo, onde o usuário cria novos projetos, adiciona e remove vídeos destes projetos. Neste módulo também são mantidos os executores e visualizadores de vídeo e animação, que são um conjunto de classes utilizadas por todos os outros módulos do protótipo.

No módulo de processamento do vídeo é mantida a tela e o conjunto de classes para o usuário manipular os vídeos do projeto e configurar os efeitos a serem aplicados nestes vídeos. Neste módulo também são mantidas as classes para segmentar os marcadores em cada quadro do vídeo.

No módulo de rastreamento dos marcadores é mantida a tela e o conjunto de classes para rastrear os marcadores em cada quadro do vídeo, isto é, definiar a trajetória de cada marcador ao longo do vídeo.

O módulo de rastreamento do modelo é responsável por manter a tela e o conjunto de classes para o usuário encaixar o modelo no primeiro quadro do vídeo, para então, efetuar o rastreamento deste modelo por todo o vídeo.

No módulo de animação é mantido o conjunto de classes para a criação do arquivo de BVH com a animação do personagem virtual e a tela para a execução e visualização desta animação.

3.3 Tela principal

Na tela principal do protótipo o usuário tem opção de criar um novo projeto e/ou abrir um projeto existente. Nesta tela o usuário mantêm os vídeos deste projeto, que posteriormente são analisados para capturar o movimento e animar o personagem virtual. Ao lado esquerdo da fig. 3 é apresentada a tela para criar e abrir projeto e ao lado direito a tela para manter os vídeos de um projeto.

Figura 3: Tela principal do protótipo

3.4 Processamento do vídeo

Na tela de processamento do vídeo o usuario pode adicionar e/ou remover efeitos que são aplicados no vídeo, com o objetivo de realçar e segmentar os marcadores do ator. Esta etapa

envolve algum conhecimento do usuário em processamento de imagens, escolhendo os efeitos adequados a serem aplicados no vídeo.

O quadro 2 apresenta os onze efeitos implementados no protótipo que o usuário pode utilizar e configurar para segmentar os marcadores no processamento do vídeo.

A fig. 4 apresenta a tela para manipulação dos efeitos de um vídeo. Ao lado esquerdo da tela o usuário pode visualizar o vídeo original e logo abaixo o vídeo com os efeitos aplicados. Do lado direito da tela são mantidos os efeitos do vídeo e na parte inferior da tela são configuradas as propriedades de cada efeito.

Efeito	*Objetivo*
Escala de cinza	Converter a imagem para escala de cinza
Brilho e contraste	Aumentar ou diminuir a intensidade da imagem
Filtro por limiar	Segmentar a imagem em regiões de interesse
Filtro negativo	Inverter a intensidade da imagem
Filtro mediana	Remover ruídos da imagem, dando um efeito de suavização
Filtro gaussiano	Remover ruídos e detalhes da imagem, dando um efeito de suavização
Filtro laplaciano	Detectar bordas na imagem
Filtro laplaciano gaussiano	Remover ruídos e detectar bordas na imagem
Filtro sobel	Detectar bordas na imagem
Bounding-box de objeto	Detectar grupos de *pixel* conectados na imagem
Transformada de hough	Reconhecer formas geométricas na imagem

Quadro 2: Lista de efeitos do protótipo

Depois de configura dos os efeitos, o usuário requisita para o protótipo criar um vídeo com os todos os efeitos aplicados, é neste momento que o sistema segmenta os marcadores em cada quadro do vídeo, para que posteriormente possam ser rastreados pelo módulo de rastreamento de marcadores.
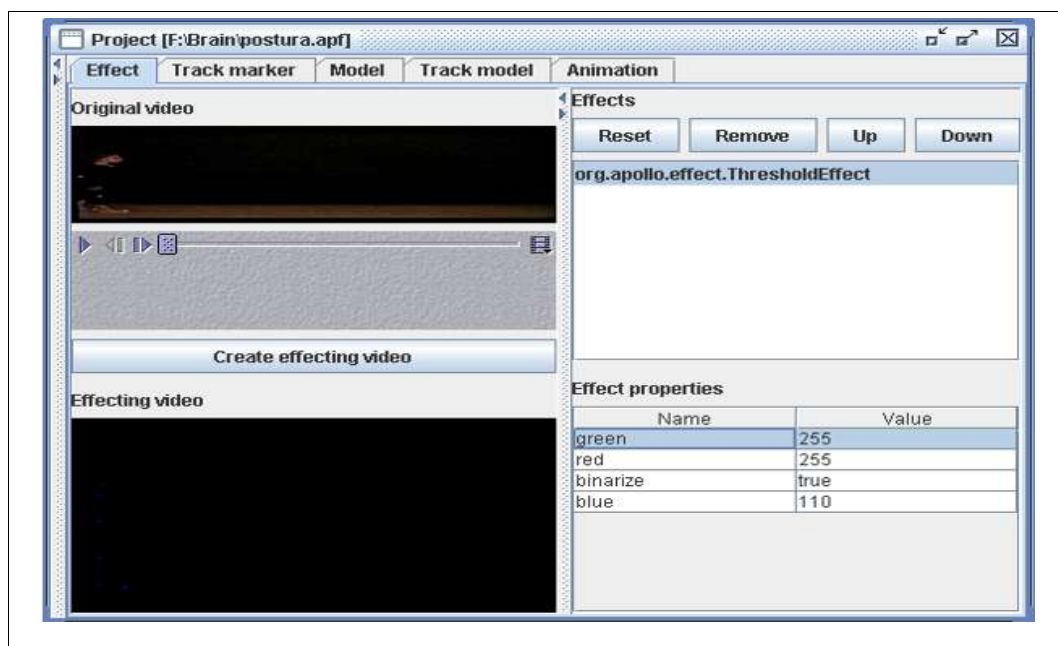
Figura 4: Tela de processamento do vídeo

3.5 Rastreamento dos marcadores

A tela de rastreamento de marcadores, apresenta na fig. 5, permite o usuário intervir nos marcadores segmentados no processamento do vídeo. Esta característica existe pois muitas vezes a qualidade do vídeo processado não é adequada, com isso, muito marcadores não são segmentados, exigindo que o usuário adicione manualmente marcadores adicionais.

Figura 5: Tela de rastreamento dos marcadores

Nesta tela o usuário requisita para o sistema rastrear os marcadores ao longo do vídeo, rastrear um marcador significa identificá-lo e informa sua posição em cada quadro do vídeo. O processo de identificação utilizado pelo sistema é de colorização, isto é, cada marcador identificado recebe uma cor única, marcadores com cor branca são considerados como não identificados.

O algoritmo de rastreamento inicialmente atribui uma cor (identificação) para os marcadores do primeiro quadro do vídeo e depois itera por cada quadro do vídeo identificando os marcadores no quadro subseqüente, isto é, ao analisar o quadro 1 o algoritmo tenta identificar os marcadores no quadro 2 e assim sucessivamente.

Para cada marcador é aplicada uma seqüencia de técnicas, e a primeira técnica é a predição de 1 quadro. Esta técnica consiste em analisar a trajetória do marcador do quadro K-1 para o quadro K, e com base nesta trajetória predizer a posição do marcador no quadro K+1. Então tenta-se achar um marcador de cor branca no quadro K+1 que esteja próximo a posição predita, caso seja encontrado, este marcador no quadro K+1 passa a ter a mesma cor do marcador do quadro N.

A segunda técnica aplicada é a do marcador mais próximo, então tenta-se achar um marcador de cor branca no quadro K+1 que esteja próximo ao marcador do quadro K, se encontrou, este marcador recebe a mesma cor.

A terceira técnica aplicada é a predição de N quadros, que para este protótipo está fixada em no máximo três quadros. O funcionamento desta técnica é idêntico ao da predição de 1 quadro, porém caso não encontre um marcador próximo a posição predita, esta técnica continua analisando até N quadros a frente. Esta técnica serve para situações em que um marcador “desapareça” por alguns quadro e então “reapareça”. Para os quadros em que o marcador esteve “desaparecido” são criados marcadores intermediários através de interpolação linear.

Os marcadores que não são identificados por nenhuma das técnicas são considerados perdidos e deixa de ser identificado no restante do vídeo.

3.6 Encaixar modelo 3D

A tela de rastreamento do modelo, apresentada na fig. 6, possibilita o usuário encaixar o modelo do personagem virtual ao ator no primeiro quadro do vídeo. Este encaixe deve ser efetuado pois as dimensões do modelo não são iguais as dimensões do ator.
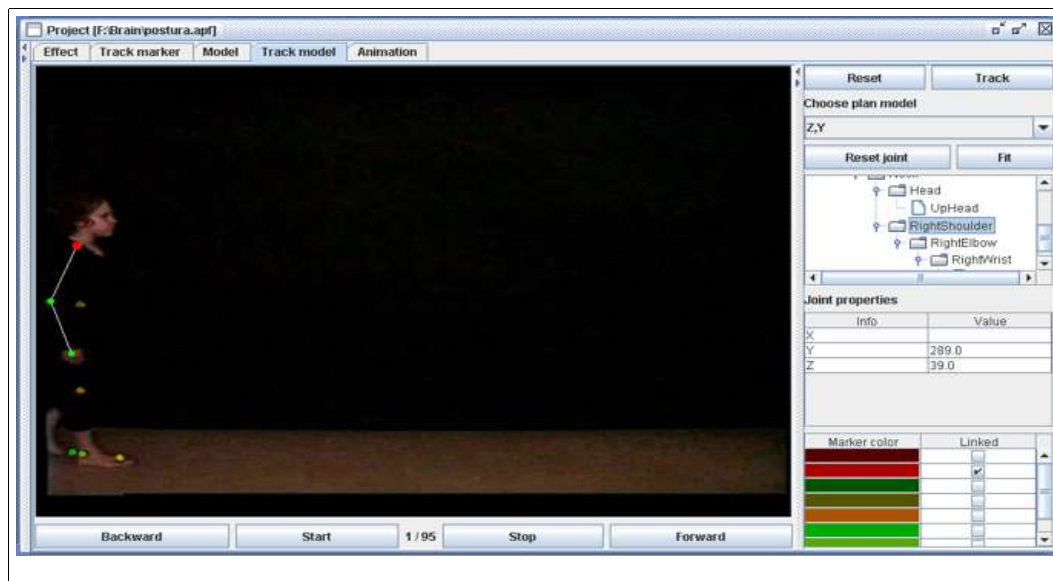
Figura 6: Tela para encaixar modelo 3D

Para encaixar o modelo o usuário informa a localização, atrabés de um clique na tela, das articulações do personagem. O usuário não precisa informar a localização de todas as articulações, pois o algoritmo efetua o encaixe automático de articulações faltantes, porém quanto mais articulações informadas, mais precisa a fica animação do personagem virtual.

Depois de informadas as articulações, o usuário requisita para o sistema encaixar o restante do modelo. Na primeira etapa do algoritmo é calculado um fator de ajuste utilizando as articulações informadas pelo usuário e o modelo original, isto é, calcular quanto o modelo original diminui ou aumentou.

O segundo passo consiste em navegar pela estrutura hierárquica do personagem e criar o novo modelo encaixado ao ator. As articulações que não são informadas pelo usuário, têm sua posição simulada através do fator de ajuste da etapa anterior.

A terceira etapa é responsável por simular as articulações que não são visualizadas pelo plano sagital. A posição destas articulações é baseada na posição da articulação do outro lado do modelo, porém a coordenada que o plano não visualiza tem seu sinal invertido. Por exemplo, o pé direito está nas coordenadas (-10, 20, 30), o pé esquerdo estará nas coordenadas (10, 20, 30), a coordenada X tem sinal invertido.

A fig. 6 apresenta a tela para encaixar o modelo, do lado direito da tela estão as articulações da estrutura hierárquica e do lado esquerdo é onde o usuário informa a posição das articulações.

3.7 Rastreamento do modelo

A tela de rastreamento do modelo, apresentada na fig. 7, permite o usuário requisitar e visualizar o rastreamento do modelo em todos os quadros do vídeo. Para o rastreamento do modelo é proposto um algoritmo flexível e escalável, onde técnicas podem ser inseridas sem

comprometer o funcionamento das técnicas já existentes. O algoritmo consiste em iterar e analisar cada quadro do vídeo, e a cada iteração é aplicado um conjunto de técnicas em ordem de confiabilidade, isto é, as técnicas que efetuam um posicionamento mais eficaz das articulações devem ser executadas antes. O presente protótipo implementa quatro técnicas para o posicionamento das articulações do modelo e estas técnicas são apresentadas na ordem de confiabilidade.
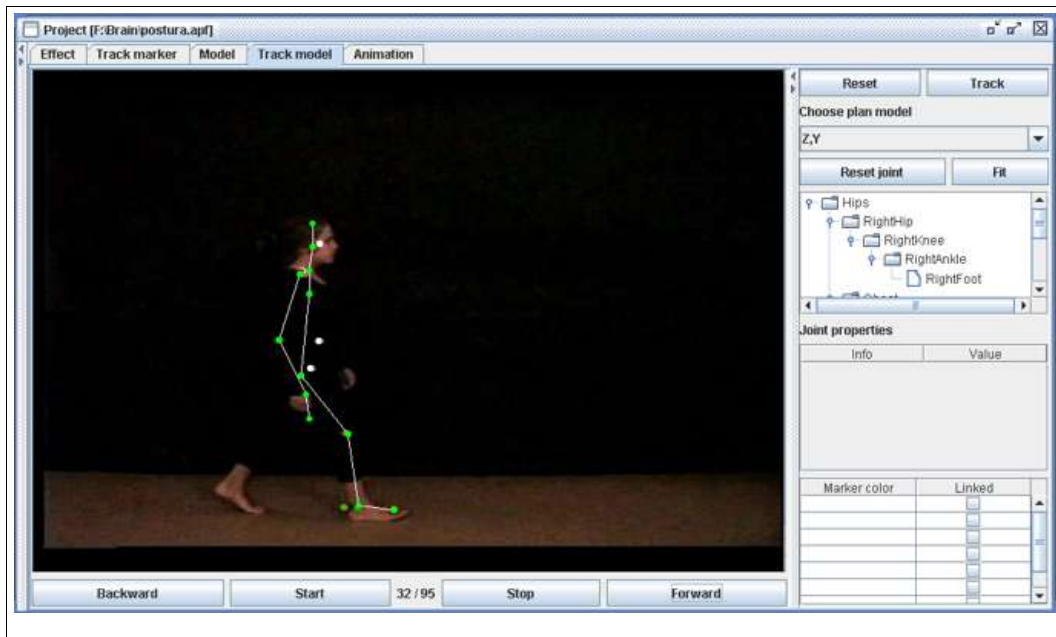
Figura 7: Tela de rastreamento do modelo

A primeira técnica aplicada para o posicionamento das articulações é a que utiliza os marcadores identificados no quadro analisado. Basicamente as articulações que tenham ligação com algum marcador e este marcador esteja identificado no quadro analisado, são posicionadas por esta técnica, em outras palavras, onde o marcador está a articulação estará. Está e a técnica que oferece o posicionamento mais preciso dentre as implementadas.

A segunda técnica aplicada para posicionar uma articulação é a que utiliza as suas articulações *pai* e *filha*. Para que esta técnica seja utilizada é necessário que as articulações *pai* e *filha* já estejam posicionadas no quadro analisado. Com base na posição das articulações *pai* e *filha*, mais a relação existente no modelo entre as três articulações, é possível calcular a posição da articulação faltante no quadro analisado.

A terceira técnica aplicada segue a mesma regra da técnica acima, só que utiliza as articulações *avô* e *pai* para efetuar o posicionamento. Conforme a regra, para que esta técnica seja utilizada, é necessário que as articulações *avô* e *pai* estejam posicionadas no quadro analisado.

A quarta técnica aplicada é a que utiliza o modelo para posicionar uma articulação, esta técnica é a mais imprecisa de todas, pois simplesmente copia a posição relativa da articulação que está no modelo, com isto, a posição relativa da articulação é sempre a mesma ao longo do vídeo.

Novas técnicas podem ser implementadas e inseridas neste algoritmo, basta que estas técnicas atendam a ordem de confiabilidade do algoritmo, por exemplo, uma técnica que utiliza as articulações *neta* e *bisneta* para o posicionamento.

3.8 Geração da animação

Na tela de animação, apresentada na fig. 8, o usuário tem a possibilidade de escolher o formato de dados que o arquivo BVH com a animação é gerado. O protótipo disponibiliza a opção de formato translacional, que armazena a posição relativa de cada articulação em relação a sua articulação *pai*. Um limitação deste formato é que a distância entre as articulações pode variar ao longo da animação.
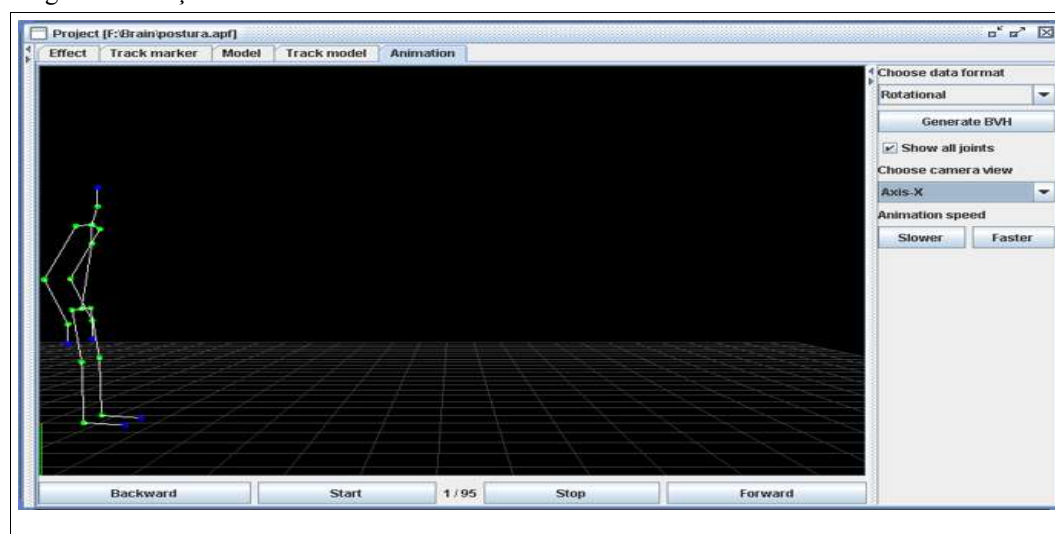
Figura 8: Tela para geração da animação

O segundo formato que o protótipo disponibiliza é o rotacional, que é o formato padrão para arquivos BVH. Neste formato é armazendo o ângulo relativo de cada articulação em relação a sua articulação *filha* para cada um dos eixos. No presente trabalho o formato rotacional está limitado apenas ao eixo X, já que os vídeos disponibilizados contêm apenas gravações do plano sagital.

Nesta tela o usuário requisita para gerar a animação, e o primeiro passo do algoritmo é gerar os dados da animação no formato definido pelo usuário. Depois são simulados os dados que não são visualizados pelo plano sagital, pois as gravações contêm apenas um lado do corpo humano, logo o outro lado deve ser simulado.

Para a simulação é utilizado a técnica de deslocamento de N quadros, que para este trabalho está fixado em 20 quadros. Para simular o dado de uma articulação não visualizada pelo plano é utilizada como base a respectiva articulação do outro lado do modelo, por exemplo, o pé esquerdo no quadro 1 tem os mesmos dados relativos ao pé direito no quadro 20.

Depois de gerados os dados do usuário e os dados simulados, o sistema gera o arquivo BVH com a animação do personagem virtual.

3.9 Execução da animação

Na tela de animação o usuário tem a possibilidade de executar a animação contida no arquivo BVH gerado. Ao lado direito da tela apresentada na fig. 8, o usuário pode selecionar uma das quatro câmeras disponibilizadas para visualização.

Na fig. 9 é apresentada uma seqüencia de quadros de uma animação, que teve como entrada um dos vídeos disponibilizados para o projeto. O formato utilizado para esta animação é o translacional.
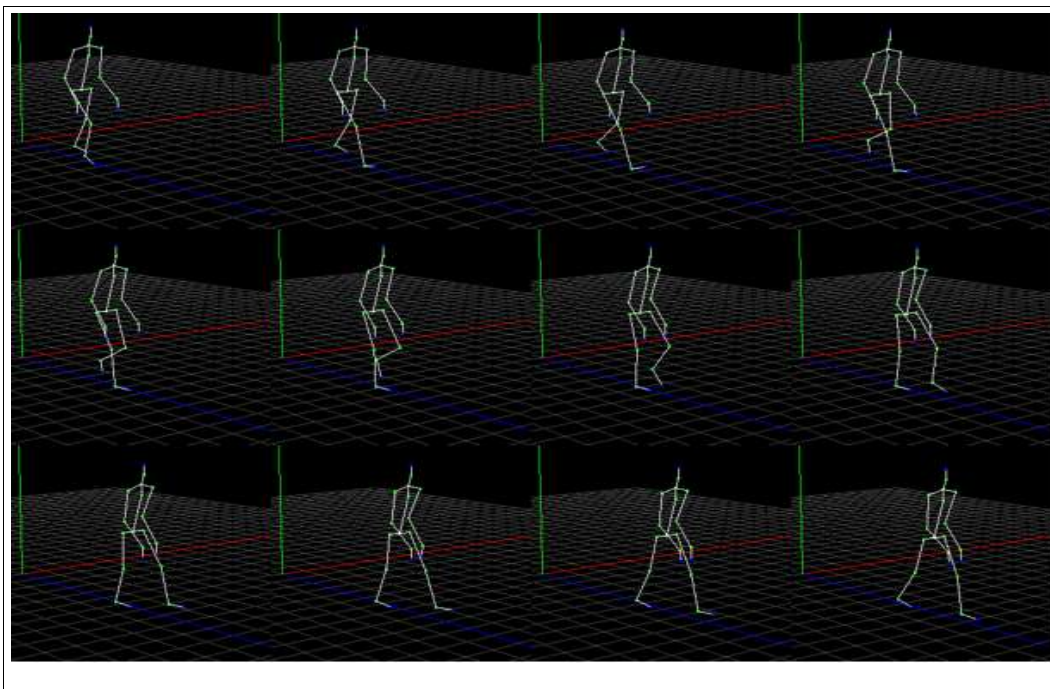
Figura 9: Sequência de quadros da animação

4 Conclusões

O presente trabalho apresentou a especificação e implementação de um protótipo de *software* para animação de um personagem virtual utilizando a captura óptica de movimento. Para isto, foi desenvolvido um módulo para processamento dos vídeos, permitindo ao usuário configurar os efeitos a serem aplicados no vídeo. O módulo de rastreamento dos marcadores foi implementado para identificar e fornecer as coordenadas 2D de cada marcador para cada quadro do vídeo. Outro módulo é o para rastreamento do modelo, permite ao usuário efetuar o encaixe do modelo ao ator do vídeo e posteriormente rastrear o modelo ao longo do vídeo. O último módulo desenvolvido é de animação, que tem por objetivo gerar e executar os arquivos BVH com animação. O desenvolvimento destes módulos soma 15.000 linhas de código, divididas em 110 classes implementadas no protótipo.

No desenvolvimento do protótipo foi utilizado um microcomputador AMD Sempron[TM] 2400+ com 1.66Ghz de freqüência e 1Gb de memória RAM. Quanto aos aspectos de desempenho do protótipo, o módulo de processamento do vídeo não executa sua tarefa em tempo real, devido ao tempo de processamento das imagens de cada quadro do vídeo. Todos os outros módulos, rastreamento dos marcadores, rastreamento do modelo e animação, obtiveram desempenho em tempo real, isto é, sem um tempo de espera para o usuário nas operações realizadas.

Como proposta de continuação do presente trabalho é sugerido a utilização de múltiplas câmeras para o rastreamento dos marcadores, utilizando processos de triangulação. Com isso, situações mais complexas podem ser capturadas e animadas pelo protótipo.

Outra sugestão é desenvolver um módulo para análise dos movimentos do ator, gerando relatórios e gráficos, atendendo assim, as necessidades do grupo de pesquisa de ergonomia da FURB.

5 Referências

AKELEY, Kurt; SEGAL, Mark. **The OpenGL graphics system**: a specification. Mountain View, 2004. Disponível em <http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf>. Acesso em: 14 jun. 2005.

APOLLO. **Motion capture and animation system**. Version 0.1. [S.l]: 2005. Disponível em <http://apollo.dev.java.net>. Acesso em: 05 out. 2005.

BAERLE, Susan Van; THINGVOLD, Jeffrey. **Motion capture file format review.** Minneapolis, [2005?]. Disponível em <http://www.gdconf.com/archives/2000/vanbaerle.doc>. Acesso em: 10 jun. 2005.

ECLIPSE. **Eclipse platform**. Version 3.1. [S.l]: 2004. Disponível em: <http://www.eclipse.org>. Acesso em: 18 maio 2005.

JAVA. **Java[tm] 2 platform standard edition**. Version 5.0. [S.l]: 2005. Disponível em <http://java.sun.com/j2se/1.5.0/index.jsp>. Acesso em: 10 jun. 2005.

JAVA3D. **Java3D API**. Version 1.3.1. [S.l]: 2003. Disponível em <http://java.sun.com/products/java-media/3D/index.jsp>. Acesso em: 10 jun. 2005.

JMF. **Java media framework API**. Version 2.1.1e. [S.l]: 2003. Disponível em <http://java.sun.com/products/java-media/jmf/index.jsp>. Acesso em: 10 jun. 2005.

JOGL. **Java bindings for OpenGL**. Version 1.1b12. [S.l]: 2005. Disponível em <http://jogl.dev.java.net>. Acesso em: 10 jun. 2005.

JUDE. **Java UML object-oriented design tool**. Version 1.5.2 community. [S.l]: 2005. Disponível em <http://jude.esm.jp/>. Acesso em: 10 jun. 2005.

PARENT, Rick. **Computer animation**: algorithms and techniques. San Francisco: Morgan Kaufmann, 2002.

SILVA, Fernando Wagner Serpa Vieira da. **Um sistema de animação baseado em movimento capturado**. 1998. 101 f. Dissertação (Mestrado em Computação Gráfica) – Laboratório de Computação Gráfica COPPE/Sistemas, Universidade Federal do Rio de Janeiro, Rio de Janeiro.

Estudo de Caso Aplicando Programação Orientada a Aspecto

Marcel Hugo (FURB/DSC)
marcel@furb.br

Marcio Carlos Grott (Totall.com S.A.)
grott@totall.com.br

Resumo. Este artigo faz um apanhado geral da tecnologia de orientação a aspecto. É feito uma introdução do tema mostrando os detalhes teóricos da composição de um aspecto, as tecnologias que dão suporte ao desenvolvimento de software e a inserção dentro da engenharia de software. Para utilizar aspectos dentro do desenvolvimento de software existem *frameworks* que fornecem uma infra-estrutura tanto no fornecimento de uma linguagem de aspecto quanto à utilização de aspetos definidos em arquivo XML. É visto com maior grau de detalhamento a linguagem de aspecto *AspectJ* que é uma linguagem baseada na sintaxe da linguagem de programação orientada a objetos Java. Para demonstrar a utilização de aspecto é elaborado um estudo de caso consistindo de um modelo de troca de informações entre sistemas heterogêneos de envio e recebimento de notificação da compra de mercadorias.

Palavras-chave: *Aspectos*, POA, *AspectJ*, *Java*, *Framework*, *Engenharia de software*.

1 Introdução

A engenharia de software e as linguagens de programação coexistem em um relacionamento de suporte mútuo. A maioria dos processos de desenvolvimento de software considera um sistema em unidades (módulos) cada vez menores. As linguagens de programação, por sua vez, fornecem mecanismos que permitem a definição de abstrações de unidades de sistemas e a composição destas de diversas formas possíveis à produção do sistema como um todo (GRADECK; LESIECKI, 2003).

Com o advento da *Object Oriented Programming* (Programação Orientada a Objetos, POO), em meados dos anos 70 com a linguagem de programação Smaltalk, inicia-se um novo paradigma de desenvolvimento de software que está presente até os tempos atuais. A POO trouxe grandes avanços para o desenvolvimento de software, permitindo a construção de sistemas mais fáceis de serem projetados, maior reusabilidade de componentes, modularidade, implementações menos complexas e redução do custo de manutenção. Muitas aplicações não estão em apenas um único módulo de código contido em um único arquivo. Aplicações são coleções de módulos (classes) que trabalham juntas para fornecer determinadas funcionalidades para um conjunto de requisitos bem definidos (GRADECKI; LESIECKI, 2003). Porém, existem certas propriedades que se pretende programar em um sistema que não são adequadamente encapsuladas em classes, seja porque se aplicam as diversas classes de um sistema simultaneamente, seja porque não pertencem à natureza intrínseca da(s) classe(s) a(s) qual(is) se aplicam. A dificuldade de modularizar alguns tipos de funcionalidades causa um acúmulo de responsabilidade adicional que o projeto da classe não previa, ocasionando um entrelaçamento da responsabilidade inicial - intrínseca da classe - com responsabilidade extra.

Devido ao entrelaçamento de responsabilidades (*crosscutting*) de uma classe surge o paradigma de programação orientada a aspectos. A implementação dos aspectos fornece subsídio a POO para tratar de tais responsabilidades extras através da decomposição funcional da classe (SOARES; BORBA, 2002). O aspecto consiste de características estruturais e/ou comportamentais que devem ser adicionadas a diversas partes do sistema, mas que são projetadas e codificadas

separadamente. Quando o sistema é compilado os aspectos combinam as responsabilidades extras, ou ortogonais, com as responsabilidades intrínsecas de cada módulo afetado, fazendo com que todas as responsabilidades apresentem as propriedades pretendidas sem comprometer a coerência de cada módulo (LADDAD, 2003).

É importante salientar que a *Aspect-Oriented Programming* (Programação Orientada a Aspectos, POA) não propõem substituir nenhum dos paradigmas já existentes, e sim, acrescentar suporte a modularização das propriedades relativas ao *crosscutting concerns* (assuntos que entrecortam o sistema). A POA não é a única proposta para suprir tal necessidade dos paradigmas em uso atualmente. Existem, por assim dizer, uma família de soluções similares, como filtros de composição, reflexão computacional, *hyper-spaces*, *Subject-Oriented Programming* (SOP), que são coletivamente denominadas de *Advanced Separations of Concerns* (técnicas avançadas de separação de assuntos) (CHAVES, 2002).

Este artigo tem por objetivo mostrar a aplicação da programação orientada a aspectos na especificação de um software orientado a objetos em camadas, implementando em *AspectJ* um aspecto do sistema – o log da leitura de arquivos XML. O artigo organiza-se em quatro seções que apresentam a orientação a aspectos, sua implementação e o estudo de caso realizado. A última seção apresenta as conclusões alcançadas.

2 Desenvolvimento de software orientado a aspectos

O advento da programação orientada a objetos tornou o desenvolvimento de sistemas mais reutilizável, flexível, com maior facilidade de manutenção e principalmente o desenvolvimento de módulos que encapsulam funcionalidades específicas do sistema. Porém o desenvolvimento orientado a objetos tem algumas limitações que são mais difíceis de serem tratadas no âmbito de objetos. Estas limitações são identificadas (OSSHER et al., 1996; OSSHER; TARR, 1999), como o entrelaçamento e espalhamento de código com diferentes propósitos, por exemplo, o entrelaçamento de código de negócio com código de apresentação, e o espalhamento de código de acesso a banco de dados em várias classes. Algumas destas limitações podem ser solucionadas com o uso de padrões de projeto (GAMA et al., 2000).

Existem algumas extensões do paradigma de orientação a objetos que tentam solucionar as suas limitações. Pode-se citar a *Aspect-Oriented Programming (*Programação Orientada a Aspectos, POA) (ELRAD; FIRMAN; BASDWER, 2001), *Subject-Oriented Programming (*Programação Orientada a Sujeito, POS) (OSSHER; TARR, 1999) e *Adaptive Programming (*Programação Adaptativa, PA). Estas técnicas visam obter uma maior modularidade de software em situações práticas onde a programação orientada a objetos e *patterns* não oferecem suporte adequado (SOARES; BORBA, 2002).

Dentro deste contexto de melhoria da qualidade e reusabilidade de código, a POA tem demonstrado ser promissora, pois procura solucionar a ineficiência em capturar algumas das principais decisões de projeto que um sistema deve implementar. Estas decisões de projeto se tornam distribuídas dentro do projeto como, por exemplo, log da aplicação, acesso a banco de dados, *cache* de objetos, tratamento de exceções, resultando num espalhamento e entrelaçamento de decisões de diferentes propósitos, aumentando a complexidade e dificultando a manutenção. Assim, a POA aumenta a modularidade separando o código que implementa funções específicas que afetam diferentes partes do sistema, chamadas preocupações ortogonais (*crosscutting concerns*). Alguns exemplos de *crosscutting concerns* é persistência, distribuição, controle de concorrência, tratamento de exceções e depuração. O aumento da modularidade implica em sistemas mais legíveis e reutilizáveis, os quais são mais facilmente projetados e mantidos.

2.1 Composição de um sistema baseado em aspectos

Um sistema que utiliza programação orientada a aspecto é composto dos seguintes componentes:

- *Linguagem de componentes*. Segundo Irwin et al. (1997), "a linguagem de componente deve permitir ao programador escrever programas que implementem as funcionalidades básicas do sistema, ao mesmo tempo em que não provêem nada a respeito do que deve ser implementado na linguagem de aspecto". As linguagens de componentes focalizam-se na resolução dos problemas encontrados durante a fase de análise preocupando-se em resolver os problemas de negócio. As linguagens de aspectos fornecem módulos com funcionalidades para resolver problemas de infra-estrutura (arquitetônico) da aplicação que podem ser reutilizados em diversos projetos com pouca ou nenhuma alteração.

- *Linguagem de aspecto*. A linguagem de aspecto deve suportar a implementação das propriedades desejadas de forma clara e concisa, fornecendo construções necessárias para que o programador crie estruturas que descrevam o comportamento dos aspectos e definam em que situações eles ocorrem (IRWIN et al., 1997);

- *Programas de componentes*: é o arquivo-fonte escrito (*source code*) em determinada linguagem de programação, por exemplo, Java, C, C++, Delphi, etc, onde o desenvolvedor codifica as regras de negócio do sistema em classes conforme definido durante a fase de análise do projeto.

- *Um ou mais programas de aspectos*: é o arquivo fonte (*source code*) codificado em uma linguagem de aspecto, como *AspectJ* (abordada adiante), *AspectC*, *AspectWerkz*, *Dynaop*. Algumas linguagens são voltadas a preocupações específicas, e neste caso devem ser utilizadas múltiplas linguagens de aspectos, uma para cada preocupação. Um exemplo de uso de múltiplas linguagens de aspectos aparece no *framework* D, onde são utilizadas duas linguagens de aspectos: *Ridl* (para distribuição) e COOL (para concorrência).

- *Combinador de aspectos*. A tarefa do combinador de aspectos (*aspect weaver*) é combinar os programas escritos em uma linguagem de componentes (as classes onde estão codificadas as regras de negócio) com os programas escritos em linguagem de aspectos (classes de aspectos). Neste trabalho será utilizada a linguagem de programação Java e o *AspectJ* como linguagem de aspecto.

Os programas de aspectos, que podem estar implementados em múltiplas linguagens de aspecto, implementam as responsabilidades ortogonais. O código do aspecto torna explícito o comportamento que será integrado ao código dos componentes e, em que contexto tal integração ocorre: são os chamados *join points (*pontos de junção), que são elementos semânticos da linguagem de componentes com os quais os programas de aspectos se coordenam. Exemplos de pontos de junção comuns são: invocações de métodos (chamadas ou recebimentos), geração de exceções, criação de objetos, entre outros.

O combinador de aspectos identifica nos componentes pontos de junção onde os aspectos se aplicam, produzindo o código final da aplicação, que programam tanto as propriedades definidas pelos componentes quanto aquelas definidas pelos aspectos. Combinadores de aspectos podem atuar em tempo de compilação ou execução. Implementações de combinadores em tempo de execução têm a possibilidade interessante de permitir a adição/exclusão de aspectos com a aplicação em pleno funcionamento. Uma visão geral de todo este processo é apresentada na figura 1.
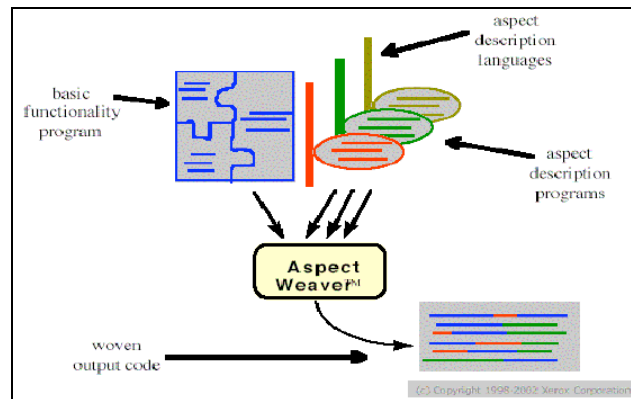
Figura 1: Composição de sistemas utilizando programação orientada a aspectos

Com este novo paradigma de desenvolvimento, POA, os analistas devem estar mais atentos em suas especificações para separar as preocupações ortogonais e os programadores devem ter em mente que as responsabilidades ortogonais devem ser deixadas para os aspectos. Assim, os programadores podem se concentrar nas reais necessidades da aplicação e o que não for inerente à resolução do problema ficará codificado em programas de aspectos. Pode-se observar que começará a existir equipes dedicadas somente com o desenvolvimento de aspectos que poderão estar presentes em diversos projetos. Enquanto, outras equipes somente estarão preocupadas com o desenvolvimento dos requisitos funcionais do sistema, utilizando os aspectos criados para compor os requisitos não funcionais do projeto.

2.2 Linguagens, propósito e características

As linguagens de aspecto de propósito específico, como o próprio nome informa, são linguagens que tratam de comportamentos específicos dentro de um sistema, como por exemplo, sincronismo, tratamento de exceções, não fornecendo suporte a qualquer outro tipo de componente. As linguagens de propósito específico possuem um nível de abstração mais alto que a linguagem de componentes para a qual foi projetada. Para garantir a utilização dos aspectos conforme foram projetadas, essas linguagens geralmente impõem alguma restrição no uso da linguagem de componente. Isto é, a linguagem de aspecto poderá impedir a utilização de determinadas palavras que são reservadas da linguagem de componentes para evitar que a linguagem de componente implemente preocupações, que seriam aspectos (STAINMACHER, 2003).

As linguagens de aspecto de propósito geral permitem a implementação de qualquer tipo de aspecto. Não é possível programar restrições junto à linguagem de componentes, pois, geralmente ambas possuem o mesmo conjunto de instruções. Como exemplo pode-se citar a linguagem *AspectJ*, que possui a linguagem Java como linguagem de componente.

A linguagem de propósito geral é mais familiar e de fácil adoção para o desenvolvimento, uma vez que poderá compartilhar o mesmo ambiente de desenvolvimento utilizado pela linguagem de componente. Como características têm:

❖ Capacidade de implementar todos os aspectos do sistema sem a necessidade de troca de ferramenta para implementar determinados aspectos não suportados pelo ambiente;

- Não é necessário o aprendizado de uma nova linguagem a cada aspecto que se implementa;
- Quando se implementa um único aspecto para uma determinada aplicação uma linguagem de aspecto específica terá um melhor desempenho e maior facilidade de desenvolvimento do aspecto. Em se tratando de uma aplicação de múltiplos aspectos a utilização de diferentes linguagens de aspecto torna-se inadequada por impor restrições à linguagem de componente a cada linguagem de aspectos que se necessite, podendo desencadear erros nos aspectos já implementados e dificultando a legibilidade da aplicação como um todo e, principalmente a manutenção da aplicação.

3 AspectJ

Aspects, os elementos básicos desta abordagem, podem alterar a estrutura estática ou dinâmica de um programa. A estrutura estática é alterada adicionando, por meio das declarações *inter-types*, membros (atributos, métodos ou construtores) a uma classe, modificando assim a hierarquia do sistema. Já a alteração em uma estrutura dinâmica de um programa ocorre em tempo de execução por meio de *join points*, os quais são selecionados por *pointcuts*, e através da adição de comportamentos (*advices*) antes ou depois dos *join points* (KISELEV, 2002).

3.1 Crosscutting elements

AspectJ utiliza extensões da linguagem de programação Java para construir novas regras para preocupações dinâmicas e estáticas. As extensões de *AspectJ* são projetadas para que programadores sintam-se a vontade enquanto estão programando os aspectos. O *AspectJ* utiliza construções próprias para os blocos de aspecto: a implementação dos aspectos é construída em blocos que formam os módulos que expressam os interesses ortogonais.

Os elementos básicos em *AspectJ* são: *join points, pointcuts, advices*, declarações *inter-types* e *aspects*. Os tópicos seguintes conceituam cada um desses elementos de *AspectJ*.

3.2 Join points

O conceito de *Join Point* é fundamental para o entendimento de *AspectJ*. O *Join Point* é qualquer ponto de execução identificado dentro de um sistema. O *AspectJ* pode operar sobre os seguintes tipos de *join points*: a) chamada de métodos, b) execução de métodos, c) chamada de construtores, d) execução de inicialização, e) execução de construtores, f) execução de inicialização estática, g) pré-inicialização de objetos, h) inicialização de objetos, i) referência a campos, j) tratamento de exceções (GRADECKI; LESIECKI, 2003).

3.3 Pointcuts

Em *AspectJ*, um aspecto normalmente define *pointcuts*, que são formados pela composição de pontos de combinação, através de combinadores lógicos. Para definir um *pointcut* utiliza-se construtores de *AspectJ* chamados designadores de *pointcuts* (*pointcuts designators*).

Um *pointcut designator* identifica o *pointcut* pelo nome ou por uma expressão. Os termos *pointcut* e *pointcut designator* são usados freqüentemente como sinônimos. Pode ser declarado um *pointcut* dentro de um aspecto, classe ou interface. Da mesma forma que atributos e métodos de classes, pode ser especificado um qualificador de acesso aos *pointcuts* (*public, private, protected* ou *default*) para restringir o acesso.

Em *AspectJ*, um *pointcut* pode ter um nome ou ser anônimo. *Pointcuts* anônimos, como classes anônimas, são definidas no lugar onde serão utilizadas, tais como parte de uma *advice*, ou

no momento da definição de outro *pointcut*. *Pointcuts* nomeados são elementos que podem ser referenciados de múltiplos lugares, aumentando a reusabilidade.

3.4 Advice

Advice é o código para ser executado em um *join point* que está sendo referenciado pelo *pointcut*. *Advice* pode ser executado antes, durante ou depois de um *join point*. O *advice* pode modificar a execução do código no *join point*, pode substituir ou passar por ele. Usando o *advice* pode-se "logar" as mensagens antes de executar o código de determinados *join points* que estão espalhados em diferentes módulos. O corpo de um *advice* é muito semelhante ao de qualquer método, encapsulando a lógica a ser executada quando um *join point* é alcançado (GRADECKI; LESIECKI, 2003).

3.5 Introduction

Introduction é um *crosscutting* estático que introduz alterações nas classes, interfaces e aspectos do sistema. Alterações estáticas em módulos não tem efeito direto no comportamento. Por exemplo, pode ser adicionado um método ou um atributo na classe.

3.6 Compile-time declaration

O *compile-time declaration* é uma declaração de *crosscutting* estático que permite adicionar alerta e erro quando se detecta certos padrões de utilização de classes. Por exemplo, pode-se declarar que é um erro fazer qualquer chamada a classes de *Abstract Window Toolkit* (AWT) em um *Enterprise Java Beans* (EJB).

3.7 Aspect

O *aspect* é a unidade central do *AspectJ* da mesma forma que a classe é a unidade central de Java. Nele estão implementadas as regras de construção de *crosscutting* dinâmico e estático. *Pointcut*, *advice*, *introduction* e *declarations* são combinados em um *aspect*. Como qualquer classe Java, os *aspects* podem conter atributos, métodos e classes internas.

Antes de começar a implementar o comportamento do *crosscutting*, precisa ser feita uma análise para identificar os *join points* e quais argumentos de comportamentos se desejam modificar, para que em seguida seja definido o novo comportamento. Para iniciar a implementação se escreve um *aspect* que serve como um modelo que contém uma implementação geral. Então, dentro do *aspect*, escrevem-se os *pointcuts* para capturar os *join points* desejados. Finalmente, se escreve os *advice* para cada *pointcut* e codifica-se a ação que se deseja executar quando o *join point* for chamado.

3.8 A lógica de compilação

O *AspectJ* oferece um conjunto de ferramentas para auxiliar na criação de aspectos, desde um compilador de aspectos, visualizadores de aspectos, *plugins* para integração com as mais populares IDE´s de desenvolvimento. Neste trabalho é utilizada a IDE Eclipse versão 3.0 com o *plugin* ADJT, desenvolvido pelo mesmo grupo de desenvolvedores da IDE, disponíveis no site da IDE (www.eclipse.org).

3.9 O compilador

O compilador é a unidade central da implementação da linguagem *AspectJ*. Ele é responsável pela combinação das classes Java com as unidades de aspectos produzindo os arquivos para serem utilizados em produção. O compilador de aspectos aceita arquivos de classes Java, unidades de

aspectos ou uma mistura de ambos. Os arquivos resultantes da compilação contêm puro *bytecodes* Java e por isto pode ser executado em qualquer máquina virtual Java.

4 Análise e desenvolvimento de um protótipo para importação de dados

Segundo Wazlawick (2004), o processo de desenvolvimento de software se divide em quatro grandes fases: análise, projeto, implementações e teste. Neste artigo será desenvolvido um ciclo de desenvolvimento que corresponde à execução das quatro etapas. Na fase de análise será feita a investigação do problema para descobrir o problema a ser resolvido.

O problema a ser resolvido corresponde a um problema do mundo real encontrado durante a análise de um B2B (*bussines to bussines*) solicitado por um cliente da empresa Totall.com S.A., que precisava que as compras de mercadorias para reposição do estoque fossem geradas e enviadas pelo sistema da Totall.com S.A. aos devidos fornecedores.

Para que pudesse dar a solução ao cliente, fez primeiramente um levantamento de requisitos avaliando as reais necessidades do cliente. Após, feito o levantamento de requisitos procedeu-se a definição dos casos de uso conforme define a especificação da UML utilizando para isto a ferramenta CASE Jude que podem ser visto na Figura 2.

Os casos de uso encontrados foram modelados para que fosse possível ter uma melhor visualização da interação do usuário com o protótipo, conforme apresentados na Figura 2.
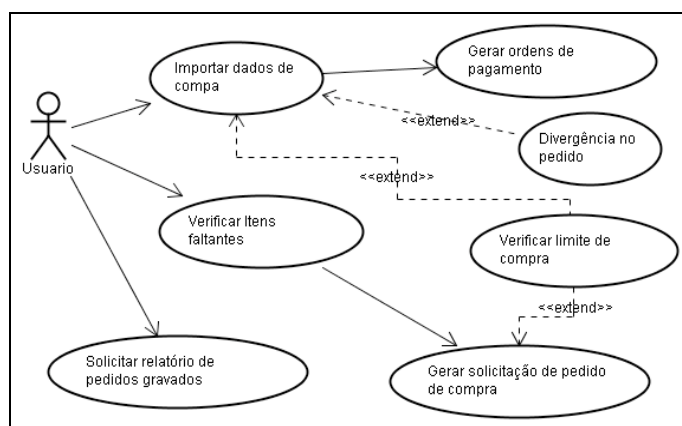
Figura 2: Modelo UML dos casos de uso

Para melhor organizar e visualizar os casos de uso se criou uma tabela que contém as informações sobre os casos e uma referência aos requisitos que compõem o caso. Na **Tabela 1** é apresentada à descrição dos casos de uso que compõem o protótipo.

Tabela 1: Listagem dos casos de uso

Nome	Atores	Descrição
Importar dados de compra	Usuário	O usuário, ao receber o arquivo referente à compra de mercadorias para o seu estabelecimento, inicia a importação dos dados da compra. Os arquivos já têm um formato predefinido entre o fornecedor e o lojista, sendo definido em formato XML, que após processado dará origem às ordens de pagamento a serem feitas ao fornecedor pelo departamento financeiro da loja. O sistema deverá fazer à baixa do pedido de compra gerado para o fornecedor informando que o pedido está concluído.
Verificar Itens faltantes	Usuário	O usuário uma vez por dia verificará quais os produtos que necessitará comprar. Caso existam produtos, o sistema deverá gerar os pedidos de compra.
Solicitar relatório	Usuário	O usuário ao final da importação poderá solicitar um relatório com o(s) pedido(s) que foram

de pedidos gravados		recebidos (importados) do fornecedor. A geração do relatório poderá ter alguns filtros que são: data, fornecedor, número do pedido.
Gerar ordens de pagamento	Usuário	Após feita a importação das compras deverá ser feita à geração da ordem de pagamento ao fornecedor. A ordem de pagamento gerada deverá notificar, por e-mail, o departamento financeiro da loja que em determinada data deverá ser feito o pagamento da compra.
Gerar solicitação de pedido de compra ao fornecedor	Usuário	As gerações dos pedidos de compras são feitos pelo usuário que depois de conferido o relatório de necessidades de mercadorias, deverá encaminhar aos fornecedores dos produtos o pedido de compra. Somente existirá um fornecedor por pedido de compra.

Encerrada a fase de levantamento de requisitos e definição dos casos de uso pode-se desenvolver o modelo da camada de domínio do protótipo. Este modelo visa produzir uma solução para o problema identificado pela análise, sem nenhuma referência às classes de aspecto bem com a sua interação dentro do protótipo (**Figura 3**).

Após definidas as classes de negócios contemplando toda a estrutura necessária para resolver os requisitos levantados no cliente e definidos nos diagramas de use-case verificou-se que havia classes que não faziam parte do problema a ser resolvido e que eram de extrema importância. Desta forma iniciou-se o levantamento destas classes e ao final do levantamento das classes faltantes concluiu-se eram classes que constituíam os aspectos da aplicação. Estes aspectos são demonstrados nos diagramas de classes com o estereótipo "Aspect" e na cor amarela.
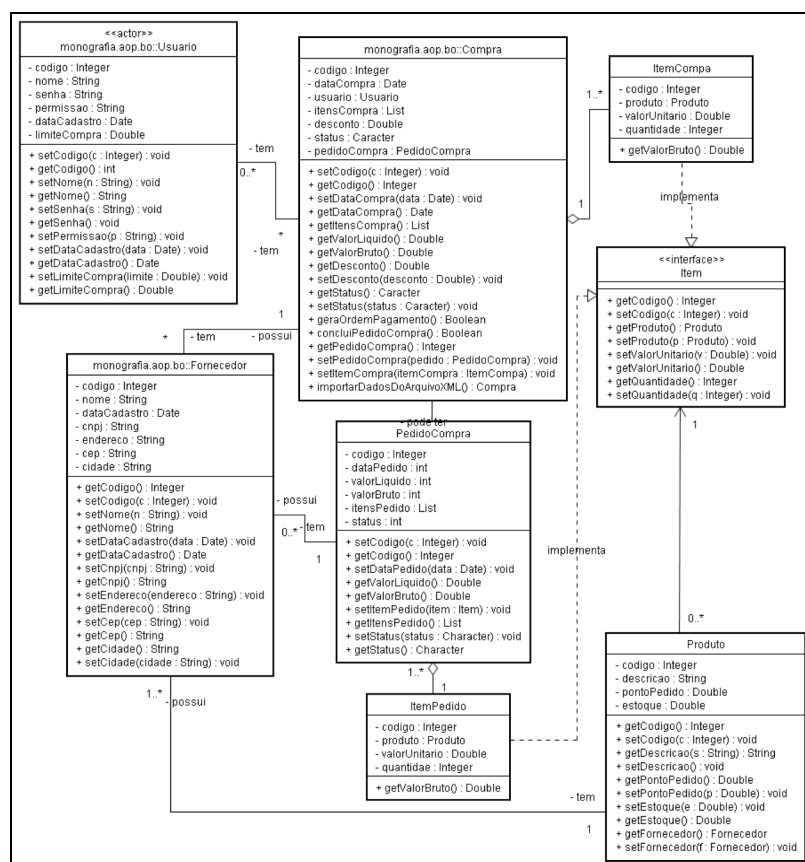
Figura 3: Diagrama de classe do protótipo

Na **Figura 4**, se tem o diagrama de classe que faz a importação dos dados oriundos do fornecedor referente à compra feita. Neste diagrama é colocado um log das operações de importação dos pedidos. Para que não seja necessário configurar um log para cada classe deste pacote, utiliza-se um aspecto que faz um log dos métodos das classes desejadas. O aspecto responsável pelo log é nomeado de AspectLog.
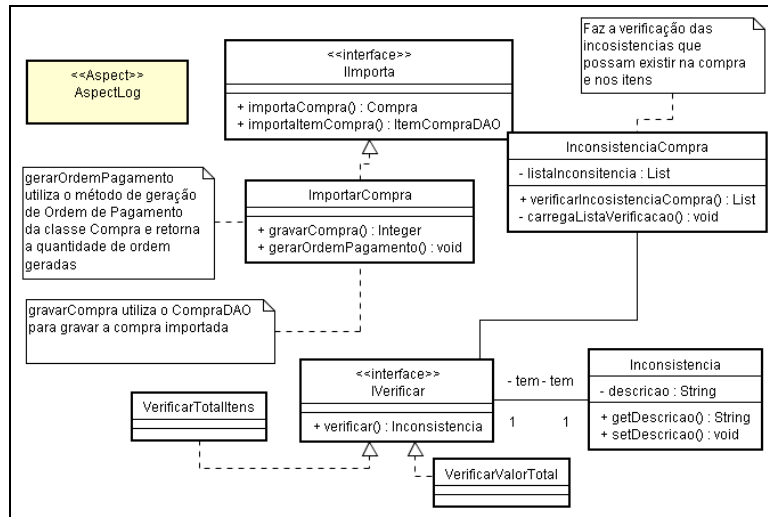
Figura 4: Diagrama de classe para importação dos pedidos

Na **Figura 5** tem-se o diagrama de classe utilizado para fazer o log das operações do protótipo definido em uma classe de aspecto. A classe de aspecto chamada de LogAspecto é utilizada para fazer gravação e leitura de arquivo(s) de log da classe GravarLog, responsável por fazer a persistência dos dados recebidos em um arquivo.
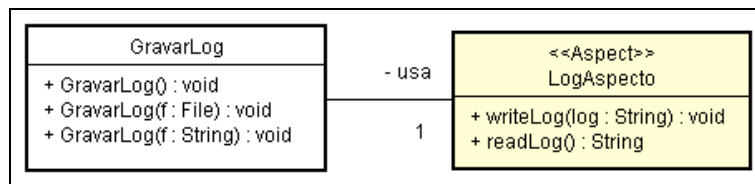
Figura 5: Diagrama de classe do aspecto que registra o log do sistema

4.1 Codificação

Com a análise do estudo de caso encerrada e elaborado os diagramas de classes pode ser feita à codificação. A codificação utiliza a linguagem de programação Java da Sun Microsystem e o *framework* de aspectagem *AspectJ* versão 1.2 desenvolvido por um consórcio de empresas e abrigado no projeto principal da IDE Eclipse.

Este protótipo está divido em duas camadas denominadas de camada de acesso a uma fonte de dados, que pode ser um banco de dados relacional, orientado a objeto, arquivos simples ou arquivo XML, e a camada de regras de negócio onde se tem os objetos que tratam da regra de negócio da aplicação. A camada de interface não será desenvolvida, pois não é o foco principal do

trabalho. Existe ainda uma camada de objeto paralela a estas duas que compreende aos objetos que fazem o transporte de dados entre as camadas.

No **Quadro 1**, apresenta-se um aspecto de log de todas as operações que serão realizadas com as classes que pertencem ao pacote `monografia`.

```
package monografia.aop.aop;
import java.text.*;
import java.util.Date;
public aspect LogAspecto {
    private    static    final    DateFormat    horaFmt    =    new
SimpleDateFormat("HH:mm:ss''S");
    pointcut recepcao() :
        execution(* monografia..*.*(..))
        && !within(Log);
    before() : recepcao() {
        System.out.println(getTimestamp() + " Iniciando "
                + thisJoinPoint.getSignature() +":" );
        Object[] arg = thisJoinPoint.getArgs();
        for(int i=0;i<arg.length;i++){
            System.out.print("arg:" + arg[i].toString() + "");
        }
    }
    after() : recepcao() {
        System.out.println(getTimestamp() + " Concluindo "
                + thisJoinPoint.getSignature()  );
    }
    public String getTimestamp() {
        return "[" + horaFmt.format(new Date()) + "]";
    }
}
```

Quadro 1: Aspecto responsável por fazer o log da aplicação

No **Quadro 2**, apresenta-se o código necessário para fazer a execução da classe de `Compra`, encontrada no diagrama de classes da **Figura 3**, que utiliza o aspecto de log para gravar as suas operações.

```
package monografia.aop.test;
import java.util.Date;
import monografia.aop.vo.Compra;
import monografia.aop.vo.Fornecedor;
import monografia.aop.vo.ItemCompra;
import monografia.aop.vo.Produto;
public class CompraTest {
    public static void main(String[] args) {
        Compra compra = new Compra();
        compra.setCodigo(Integer.valueOf(1));
        compra.setDataCompra(new Date());
        compra.setDesconto(Double.valueOf(10));
        compra.setFornecedor(new Fornecedor("02882917945"));
        compra.setStatus(Character.valueOf('C'));
        //   método   que   realiza   a   importação   dos   dados   do
arquivo XML
        compra.importarDadosDoArquivoXML();

        System.out.println("done");
    }}
```

Quadro 2: Trecho de código utilizado para demonstrar a utilização do aspecto de log.

Para demonstrar a execução do aspecto no **Quadro 3**, é capturado um trecho do resultado da execução do aspecto de log `LogAspecto`.

```
[21:04:10'484] Iniciando void
monografia.aop.test.TesteCompra.main(String[]):
arg:[Ljava.lang.String;@8813f2[21:04:10'609] Iniciando void
monografia.aop.vo.Compra.setCodigo(Integer):
arg:1[21:04:10'609] Concluindo void
monografia.aop.vo.Compra.setCodigo(Integer)
[21:04:10'609] Iniciando void
monografia.aop.vo.Compra.setDataCompra(Date):
arg:Sun May 08 21:04:10 BRT 2005[21:04:10'625] Concluindo void
monografia.aop.vo.Compra.setDataCompra(Date)
[21:04:10'625] Iniciando void
monografia.aop.vo.Compra.setDesconto(Double):
arg:10.0[21:04:10'625] Concluindo void
monografia.aop.vo.Compra.setDesconto(Double)
[21:04:10'625] Iniciando void
monografia.aop.vo.Fornecedor.setCnpj(String):
arg:02882917945[21:04:10'625] Concluindo void
monografia.aop.vo.Fornecedor.setCnpj(String)
[21:04:10'625] Iniciando void
monografia.aop.vo.Compra.setFornecedor(Fornecedor):
arg:monografia.aop.vo.Fornecedor@de6f34[21:04:10'625] Concluindo void
monografia.aop.vo.Compra.setFornecedor(Fornecedor)
[21:04:10'625] Iniciando void
monografia.aop.vo.Compra.setStatus(Character):
arg:C[21:04:10'625] Concluindo void
monografia.aop.vo.Compra.setStatus(Character)
done
[21:04:10'625] Concluindo void
monografia.aop.test.TesteCompra.main(String[])
```

Quadro 3: Listagem da execução do aspecto de log.

5 Conclusão

A programação orientada a aspecto prova ser uma nova abordagem de desenvolvimento de software caracterizada pela capacidade de abstração de módulos que não fazem parte da análise do problema solicitado pelo cliente. Outra grande melhoria que traz é a capacidade de reutilização de grande parte dos módulos desenvolvidos, sendo a reutilização algo que se almeja cada vez mais no desenvolvimento orientado a aspecto.

Durante o desenvolvimento do protótipo pôde-se notar a facilidade com que foi implementado um aspecto de log utilizado por todas as demais classes sem a necessidade de fazer a programação manual em cada classe afetada por este aspecto. Por outro lado, houve certa dificuldade no desenvolvimento de aspectos no sentido de realizar a análise do problema sem se preocupar com os aspectos ortogonais da aplicação, ou seja, ao longo da análise sempre temos em mente todo o arcabouço de recursos necessários para resolução do problema, que nem sempre faz parte do domínio do problema e sim de sua implementação. Isto faz com que não consigamos facilmente abstrair todos estes recursos e pensar somente na resolução do problema proposto.

Foi possível verificar que a programação orientada a aspecto traz de forma transparente e com resultados imediatos a reutilização de unidades de códigos. O aspecto de log desenvolvido para fazer o log de uma importação de dados de compras do fornecedor poderá ser utilizado em todas as classes de módulo como também, em vários outros sistemas sem a necessidade de recodificação de um aspecto de log.

Referências

CHAVES, Rafael Alves. **Aspectos e Midlware**. 2002. 50f. Trabalho Individual submetido à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Mestre em Ciências da Computação, Florianópolis.

ELRAD, T., FILMAN, R. E., and BADER, A. (2001). **Aspect–oriented programming. Communications of the ACM**, 44 (10):29–32.

GAMMA, Erich, et al. **Padrões de Projeto: soluções reutilizáveis de software orientado a objetos**. Trad. Luiz A. Meirelles Salgado. Porto Alegre: Bookman, 2000.

GRADECK, Joe, LESIECKI, Nicolas. **Mastering AspectJ: aspect-oriented programming in Java**. Indianopolis, Indiana. Wiley, 2003, p. 453.

IRWIN, John, et al. **Aspect – Oriented Programming**. Proceeding of ECOOP'97, Finland: Springer - Verlag, 1997.

KISELEV, I. **Aspect – Oriented Programming with AspectJ**. Ed. Sams Publishing, 2002.

LADDAD, Ramnivas. **AspectJ in Action: Practical Aspect-Oriented Programming**. Ed. Mannig, Canada, [2003].

OSSHER, H., et al (1996). **Specifying suject–oriented composition**. TAPOS, 2 (3):179–202. Special Issue on Subjectivity in OO Systems.

OSSHER, H. and TARR, P. (1999). **Using subject–oriented programming to overcome common problems in object–oriented software development/evolution**. In International Conference on Software Engineering, ICSE'99, pages 698–688. ACM, 1999.

SOARES, S. and BORBA, P. (2002). **Progressive implementation with aspect–oriented programming**. In Verlag, S., editor, The 12th Workshop for PhD Students in Object-Oriented Systems, Malaga, Spain. To appear.

STAINMACHER, Igor Fábio. **Estudo de Princípios para Modelagem Orientada a Aspectos**. 2003. 70 f. Trabalho de conclusão apresentado ao Curso de Ciências da Computação, do Centro de Tecnologia da Universidade Estadual de Maringá, Maringá.

WAZLAZWICK, Raul Sidnei. **Análise de Projeto de Sistemas de Informação Orientados a Objetos**. Rio de Janeiro: Elsevier, 2004.

Ferramenta para Geração de Código a partir da Especialização do Diagrama de Classes

Alexandro Deschamps (Ápice)
alexandro@apicesoft.com

Everaldo Artur Grahl (FURB/DSC)
egrahl@furb.br

Resumo. Uma das grandes preocupações das organizações desenvolvedoras de softwares é a busca da qualidade de software. Uma das ênfases abordadas pelos Sistemas de Garantia da Qualidade é a definição do processo de desenvolvimento de software das organizações. Para obter um processo de desenvolvimento de software bem definido, as organizações utilizam as metodologias abordadas pela Engenharia de Software em conjunto com o uso de ferramentas que automatizem este processo. Este artigo tem a finalidade de apresentar uma ferramenta de auxílio ao processo de desenvolvimento de software que possibilita a especialização dos diagramas de classes UML (*Unified Modeling Language*) por arquitetura e idioma. A finalidade desta especialização é a geração de código através do uso de *plugins*. A ferramenta apresentada ajuda a agregar uma melhor padronização ao processo de desenvolvimento, diminui o tempo de implementação e aumenta a qualidade dos softwares desenvolvidos, tornando-os mais flexíveis diante das futuras mudanças tecnológicas.

Palavras-chave: UML, Diagrama de classes, Geração de código, Internacionalização.

1 Introdução

Nos dias atuais os principais requisitos exigidos pelo mercado em relação às organizações desenvolvedoras de software são: a qualidade, o tempo e a tecnologia utilizada. O responsável por garantir os requisitos citados é processo de desenvolvimento de software, sendo que este processo é dividido em etapas e se utiliza de diferentes metodologias de desenvolvimento e arquiteturas como linguagens de programação, banco de dados, etc. Um dos aspectos abordados pelo processo de desenvolvimento de software e de suma importância para o mesmo é a codificação. Uma codificação construída sem o uso de metodologias e padrões se torna um agravante para os quesitos tempo e qualidade, dificultando também a manutenção do software em questão. Outro fator em destaque é a constante mudança tecnológica, que na maioria das vezes, torna a codificação dos softwares voláteis, forçando as organizações a reescreverem os mesmos.

A UML (*Unified Modeling Language*) é atualmente a linguagem para documentação e modelagem de software mais difundida entre os desenvolvedores. Através dos modelos e diagramas propostos pela UML pode-se obter diferentes visões de um software, dentre eles, um diagrama que recebe um destaque especial é o diagrama de classes. O diagrama de classes é importante não só para visualização, especificação e para a documentação de modelos estruturais, mas também para a construção de sistemas executáveis por intermédio da Engenharia da Produção (*Forward Engineering*) (BOOCH; RUMBAUGH; JACOBSON, 2000, p. 104). A Engenharia da Produção é o processo de transformar um modelo em código (BOOCH; RUMBAUGH; JACOBSON, 2000, p. 112). Muitas das ferramentas CASE (*Computer Aided Software Engineering*) existentes no mercado contemplam o processo de Engenharia da Produção, mas geralmente seus recursos são extremamente limitados e não permitem ao desenvolvedor realizar a geração de código de uma forma mais refinada, não atendendo desta forma também as particularidades dos diferentes processos de desenvolvimento de software.

Um exemplo de refinamento que se torna evidente quando se leva em consideração a internacionalização de software é a necessidade da criação de softwares que suportem diferentes idiomas. Utilizando somente a especificação do diagrama de classes proposto pela UML isto não é possível. Outro refinamento que se pode citar é a utilização de um mesmo modelo de classes para geração de código em diferentes arquiteturas.

Dentro deste contexto o objetivo deste artigo é apresentar o desenvolvimento de uma ferramenta de auxílio ao processo de desenvolvimento que permita a especialização das informações contidas em diagramas de classes UML por arquitetura e idioma, visando enriquecer desta maneira o processo de Engenharia da Produção.

A geração de código proposta ocorre através da utilização de *plugins*. Os *plugins* de geração são módulos DLL (*Dynamic Link Library*) encarregados de montar a sintaxe do código, estando os mesmos ligados a uma determinada arquitetura. As informações contidas nos diagramas de classes e suas especializações são disponibilizadas através de uma interface padrão utilizada para captação de dados pelos *plugins*.

O artigo foi organizado da seguinte forma: na seção 2 é realizada a descrição da ferramenta através da apresentação das suas estruturas e processos. Na seção 3 é apresentado um estudo de caso que visa através de exemplos apresentar à aplicabilidade da ferramenta para a geração de código. Na seção 4 são apresentadas as conclusões e a importância da ferramenta para o processo de desenvolvimento de software.

2 A ferramenta

A constante mudança tecnológica na maioria das vezes força as organizações a reescreverem seus softwares, com base neste problema e na experiência em desenvolvimento de softwares o autor através de estudos e aplicação de conceitos de Engenharia de Software buscou através da criação de uma ferramenta amenizar este e outros problemas comuns no processo de desenvolvimento de software. A ferramenta criada utiliza como base tecnológica a plataforma Microsoft .NET e tem o objetivo de auxiliar o processo de desenvolvimento de software através do enriquecimento das informações contidas em diagramas de classes UML (*Unified Modeling Language*) para geração de código. O enriquecimento proposto ocorre através da especialização dos elementos dos diagramas de classes por arquitetura e idioma.

Apesar de possuir uma notação rica, os diagramas de classes não apresentam características que possibilitem uma geração de código mais refinada, como exemplo disto pode se citar as características de internacionalização de um software. Outro problema é a forte ligação do diagrama de classes com uma tecnologia em particular.

Dentro deste contexto é possível através da utilização da ferramenta agregar maior flexibilidade e personalização a este processo. Isto ocorre através da estrutura de especialização proposta, está estrutura permite adicionar aos diagramas de classes novas características. A estrutura de especialização permite a criação de características por arquitetura e idioma, prevendo também as diferentes categorias de classes utilizadas na construção de um software. Com a utilização dos diagramas de classes em conjunto com as estruturas de especialização obtém-se como resultado um repositório rico e padronizado de informações para geração de código. Maiores detalhes sobre a estrutura de especialização pode se obter em Deschamps (2005).

Com base nas diferentes necessidades e particularidades encontradas nos processos de desenvolvimento de softwares das organizações, a geração de código proposta pela ferramenta visa a construção de *plugins* (pacotes) que contenham a lógica da geração de código conforme a necessidade de cada organização. Visando desta forma o processo de desenvolvimento de cada organização e não somente as tecnologias existentes no mercado.

2.1 Diagrama de atividades

O diagrama de atividades apresentado na Figura 1 demonstra uma visão geral do fluxo das informações e processos necessários para geração de código através da ferramenta. Mais informações sobre a especificação da ferramenta como o diagrama de casos de uso e o diagrama de classes podem ser vistos em Deschamps (2005).
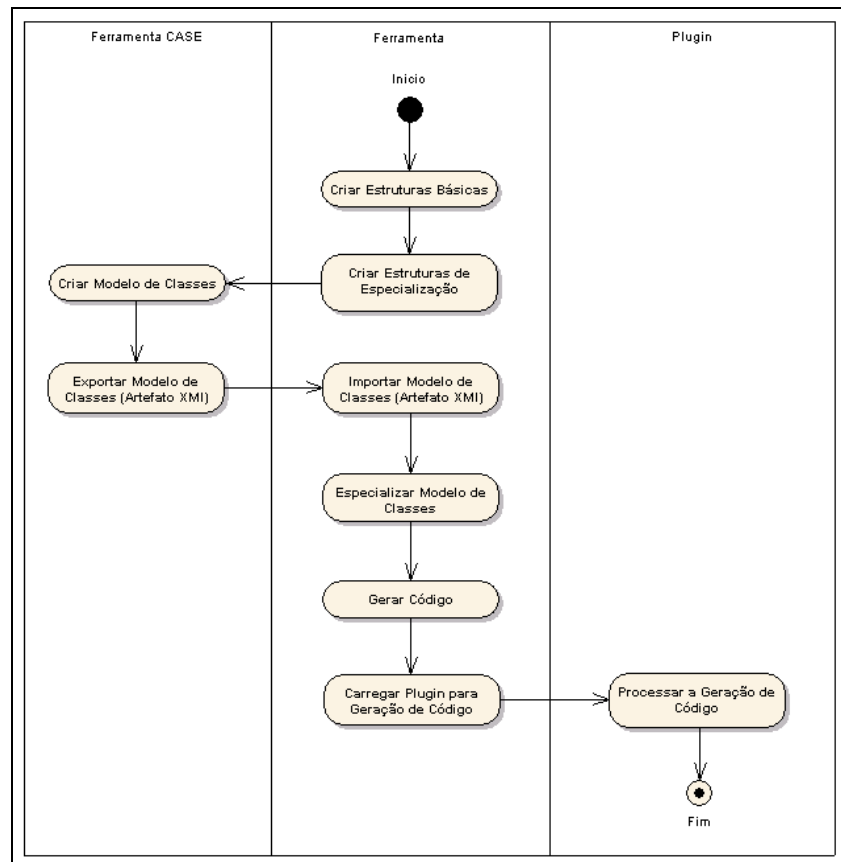
Figura 1: Diagrama de atividades

2.2 Estrutura de especialização

A estrutura de especialização visa agregar a cada elemento do diagrama de classes (Pacotes, Interfaces, Classes, Atributos, Operações, Parâmetros e Relacionamentos) novas características ou propriedades.

2.2.1 Categorias de especialização e suas propriedades

Uma categoria de especialização é composta por um conjunto de propriedades e tem o objetivo de manter um agrupamento das mesmas por categoria. Um exemplo de categoria de especialização pode ser apresentado através das diferentes categorias de "classes" propostas pela Engenharia de Software, onde as mesmas podem ser divididas em classes de entidades, classes de

controle e classes de fronteira, sendo que cada uma das categorias citadas apresentam características ou propriedades de caráter particular.

Uma propriedade de especialização possui nove tipos de dados, sendo eles: Frase, Texto, Número Inteiro, Número Decimal, Lógico, Data, Hora, Lista de Seleção e Lista de Múltipla Seleção. Ao criar uma propriedade de especialização o desenvolvedor realiza a escolha do tipo de dado que a propriedade deve armazenar. Os tipos de dados apresentados visam oferecer ao desenvolvedor maior flexibilidade e organização na criação das propriedades de especialização. Como exemplo pode se citar o tipo de dado "Lógico" que pode ser utilizado na especialização de um determinado "atributo", indicando se o mesmo é visível ou não, outro exemplo é o tipo de dado "Texto" que pode ser utilizado para documentar os elementos do diagrama de classes.

As propriedades de especialização possuem quatro tipos de visibilidade, cada tipo pode estar relacionado com uma arquitetura e/ou idioma. A Tabela 1 apresenta os tipos de visibilidade juntamente com seus níveis de dependência.

Tabela 1: Tipos de visibilidade das propriedades de especialização

Tipos de visibilidade	Dependente de arquitetura	Dependente de idioma
Genérica		
Arquitetura	X	
Idioma		X
Arquitetura e Idioma	X	X

O tipo de visibilidade "Genérica" oferece ao usuário a possibilidade da criação de propriedades que sejam completamente independentes de arquitetura e idioma, a utilização deste tipo de propriedade proporciona uma considerável reutilização das especificações armazenadas no repositório da ferramenta. Um exemplo para este tipo de propriedade pode ser apresentado através de um indicativo que informa se um determinado "atributo" aceita valor nulo ou não, nota-se que seu valor se manterá o mesmo independente da arquitetura ou idioma utilizado.

O tipo de visibilidade "Arquitetura" tem o objetivo de agrupar propriedades que pertençam a uma arquitetura em particular. Um exemplo para este tipo de visibilidade pode ser apresentado através da especialização de uma "classe" de interface para arquitetura Delphi, a mesma possui uma propriedade denominada "HelpContext" não contemplada pelas demais arquiteturas.

O tipo de visibilidade "Idioma" tem o objetivo de agrupar propriedades que pertençam a um determinado idioma em particular. Um exemplo para este tipo de visibilidade pode ser apresentado através da criação de uma descrição ou rótulo para um determinado atributo, havendo desta forma a necessidade de especificá-lo para cada um dos idiomas utilizados.

O tipo de visibilidade "Arquitetura e Idioma" é o menos comum, a criação do mesmo se aplica as propriedades de especialização que pertencem a uma determinada arquitetura e idioma. Um exemplo de aplicação deste tipo de visibilidade é a documentação de uma determinada "operação", caso esta seja implementada em diferentes arquiteturas e necessite ser documentada em vários idiomas.

A estrutura de especialização segue o modelo de herança semelhante ao utilizado pela orientação a objetos. A utilização do modelo de herança permite a criação de categorias de especializações descendentes, sendo que as mesmas herdam todas as propriedades da sua categoria de especialização ancestral.

As propriedades de especialização herdadas de uma categoria de especialização ancestral podem ser sobrescritas para a categoria de especialização descendente através da atribuição de um novo valor para a mesma. A Tabela 2 apresenta as possíveis descendências das propriedades de especialização por tipo de visibilidade.

Tabela 2: Possíveis descendências por tipo de visibilidade

Tipos de visibilidade ancestrais	Tipos de visibilidade descendentes
Genérica	Genérica Arquitetura Idioma Arquitetura e Idioma
Arquitetura	Arquitetura Arquitetura e Idioma
Idioma	Idioma Arquitetura e Idioma
Arquitetura e Idioma	Arquitetura e Idioma

2.2.2 Importação e utilização das categorias de especialização

A importação do diagrama de classes ocorre através da utilização da tecnologia XMI (*XML Metadata Interchange*), o XMI é um formato padrão de intercâmbio de meta dados entre aplicações, repositórios e ferramentas baseado em XML (*Extensible Markup Language*). Entre os benefícios trazidos com a utilização desse padrão estão a facilidade de implementação, por parte das organizações, nos produtos atuais, quebra da barreira entre ferramentas, repositórios e aplicações incompatíveis e o fato de trabalhar com tecnologias que já são padrões da indústria como XML e UML (OBJECT MANAGEMENT GROUP, 2002)

As categorias de especialização, assim como cada elemento do digrama de classes, possuem um "estereótipo". Este mecanismo é utilizado para estender o significado de um determinado elemento de um diagrama (BEZERRA, 2003, p. 41). Ao realizar a importação de um diagrama de classes a ferramenta verifica para cada elemento do diagrama a existência de uma categoria de especialização com estereótipo idêntico ao informado para o elemento na fase de modelagem. Verificada a existência a ferramenta cria uma nova especialização para o item importado, utilizando como ancestral a categoria de especialização encontrada. Desta forma o desenvolvedor tem a possibilidade de sobrescrever os valores das propriedades para os elementos dos digramas de classes importados, informando as características particulares de cada um.

2.3 Geração de código

A ferramenta não segue o modelo de pacote fechado, uma das principais características da mesma é a flexibilidade oferecida aos desenvolvedores. Através da utilização da ferramenta uma organização pode adaptar o seu processo de Engenharia da Produção (*Forward Engineering*) conforme as necessidades e particularidades da mesma.

Ao contrário das soluções disponíveis no mercado, a ferramenta não trabalha com a utilização de *scripts* para implementação das rotinas de geração. As rotinas com a lógica para geração de código são construídas através de *plugins*, módulos DLL (*Dynamic Link Library*) compilados através da plataforma Microsoft .NET. Um dos principais fatores para a escolha desta plataforma foi a possibilidade da utilização de diferentes linguagens de programação para construção dos mesmos. Desta forma as organizações que desejarem criar os seus *plugins* podem realizar a implementação dos mesmos utilizando a linguagem de programação mais próxima da sua realidade.

Os principais motivos da utilização de *plugins* compilados ao invés de *scripts* são: o considerável aumento de velocidade na geração e o ganho de recursos para construção dos mesmos, sendo possível desta forma utilizar todos os subsídios fornecidos pela plataforma Microsoft .NET. Dentre eles pode-se citar a orientação a objetos e a utilização de uma linguagem

de programação já conceituada no mercado como JAVA, Visual Basic, Delphi e C#, abstraindo desta forma a necessidade do estudo de uma nova tecnologia ou *script* por parte do desenvolvedor.

A utilização de diagramas de classes UML e suas especializações fornecem ao desenvolvedor um repositório rico em informações para a geração de código para diferentes arquiteturas e idiomas. A preocupação de utilizar um mesmo repositório de dados para gerar código em diferentes arquiteturas também é perceptível em Schmidt (2001) e Wehrmeister (2001).

3 Estudo de caso

Este estudo de caso tem o objetivo de apresentar um exemplo de geração de código com base em um diagrama de classes de localizações criado através da ferramenta CASE Enterprise Architect, demonstrando desta forma a capacidade de especialização da ferramenta por arquitetura e idioma. Para a especialização por arquitetura foram utilizadas as arquiteturas JAVA e PCL (*PHP Custom Library*), uma biblioteca para desenvolvimento em PHP (*Hypertext Preprocessor*) fornecida pela Ápice Engenharia de Software uma empresa de software de Blumenau / Santa Catarina. Na especialização por idioma foram utilizados os idiomas português e inglês. Outro objetivo é demonstrar a diferença entre um código gerado no padrão das ferramentas CASE (*Computer Aided Software Engineering)* existentes no mercado e um código gerado através da ferramenta apresentada pelo artigo. A Figura 2 apresenta o diagrama de classes de localizações, este diagrama possui três classes de entidade (“Pais”, “Estado” e “Municipio”).
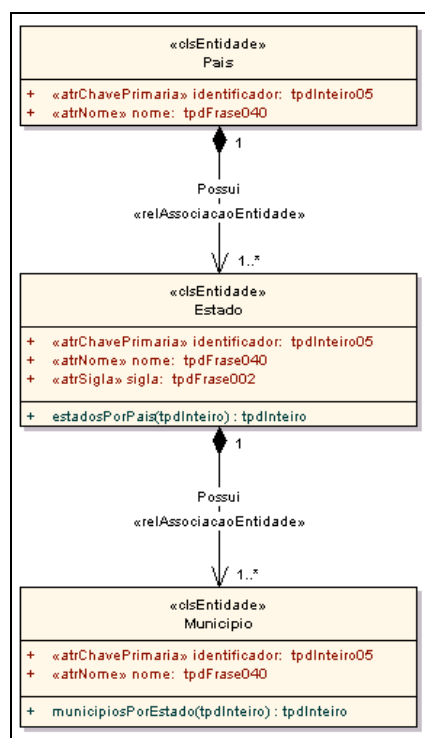
Figura 2: Diagrama de localizações

Para a apresentação deste exemplo foram construídos dois *plugins* para geração de código, um para a arquitetura JAVA e outro para a arquitetura PCL.

A seguir é apresentado um exemplo de especialização realizado para a arquitetura JAVA. A geração de código para a arquitetura JAVA tem o objetivo de gerar um código semelhante ao gerado pelas ferramentas CASE (*Computer Aided Software Engineering*) existentes no mercado. A Figura 3 apresenta a propriedade de especialização "Tipo de Dado" para a especialização do tipo de dado "Inteiro". Nota-se que este tipo de dado é utilizado pelo diagrama de localizações apresentado na Figura 2. Nota-se também que a propriedade em questão está sendo sobrescrita para a arquitetura Java com o valor "int".
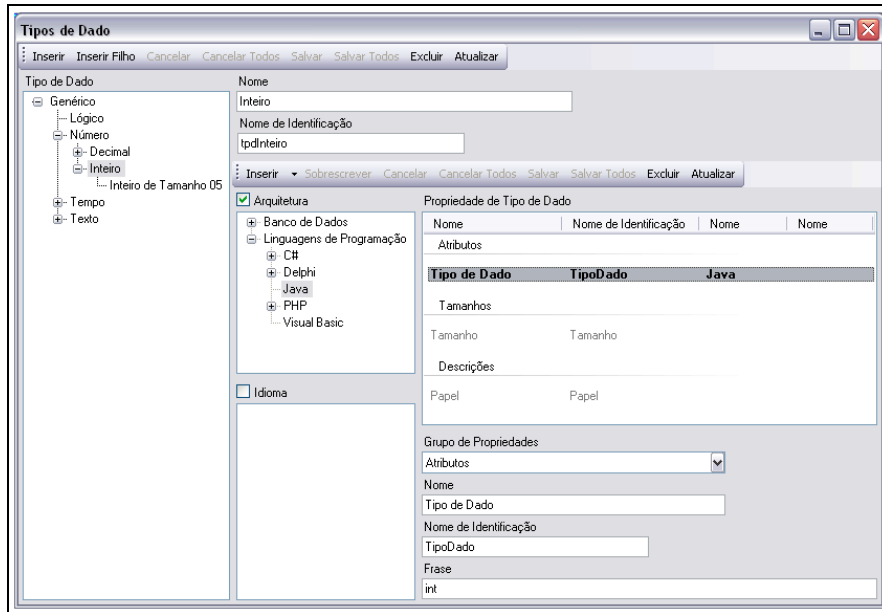
Figura 3: Exemplo da especialização do tipo de dado "Inteiro"

O Quadro 1 apresenta um trecho do código gerado para a classe "Estado". Nota-se em destaque a utilização do tipo de dado "Inteiro" anteriormente citado.

```
/*
Fonte Gerado Automaticamente
Plugin: QuelleGenerationJava
Data: 6/6/2005
Hora: 18:16
*/

public class Estado {

    private int identificador = null;
    private String nome = null;
    private String sigla = null;
    private Pais pais = null;

    public Estado() {
        super();
    }

    public int getIdentificador() {
        return this.identificador;
    }
```

Quadro 1: Trecho de código Java gerado para a classe "Estado"

A seguir é apresentado um exemplo de especialização realizado para a arquitetura PCL. A geração de código para a arquitetura PCL tem o objetivo de demonstrar uma geração de código mais completa e refinada. O principal motivo para a escolha desta arquitetura é a necessidade da criação de três subclasses (“Atributos”, “Visoes” e “Relacionamentos”) que auxiliam a classe de entidade em questão na manipulação de seus dados.

O Quadro 2 apresenta um trecho do código gerado para a subclasse “Atributos” da classe “Estado”. A subclasse “Atributos” tem o objetivo de fornecer uma lista com todos os atributos para a classe de entidade em questão. Os atributos são adicionados na lista conforme o seu tipo de dado.

```
class EstadoAtributos extends CustomEntityAttributes {

    var $identificador = NULL;
    var $nome = NULL;
    var $sigla = NULL;
    var $pais = NULL;

    function EstadoAtributos($owner) {
        $this->CustomEntityAttributes($owner);

        $this->identificador =
        $this->addAttributeInteger("identificadorEstado",
                                   "id_est",
                                   "Identificador do Estado",
                                   true,
                                   false,
                                   false,
                                   5,
                                   NULL,
                                   NULL);

        $this->nome =
        $this->addAttributeString("nomeEstado",
                                  "nm_est",
                                  "Nome do Estado",
                                  false,
                                  false,
                                  false,
                                  40,
                                  NULL,
                                  NULL);
```

Quadro 2: Trecho do código PCL gerado para a subclasse “Atributos” da classe “Estado”

As classes de entidade da arquitetura PCL também requerem para a geração de código algumas propriedades como “Nome Lógico”, “Nome Físico” e “Descrição”, que por serem de caráter particular, não são suportadas pelos diagramas de classes UML (*Unified Modeling Language*). Neste caso surge a necessidade da criação das respectivas propriedades na especialização das classes de entidade na ferramenta conforme apresentado na Figura 4.

A Figura 5 apresenta a interface de especialização dos modelos de classes. Nota-se que a mesma demonstra a especialização da classe “Estado”, por ter o mesmo estereótipo que a especialização das classes de entidades demonstrada na Figura 4. A classe “Estado” herda todas as propriedades previamente informadas para a mesma.

Também é demonstrado através da Figura 5 a atribuição de um novo valor para a propriedade “Descrição”. Este novo valor é atribuído ao sobrescrever o valor anteriormente cadastrado na especialização das classes de entidade, conforme demonstrado na Figura 4. Nota-se também que o mesmo está sendo especificado para o idioma português.
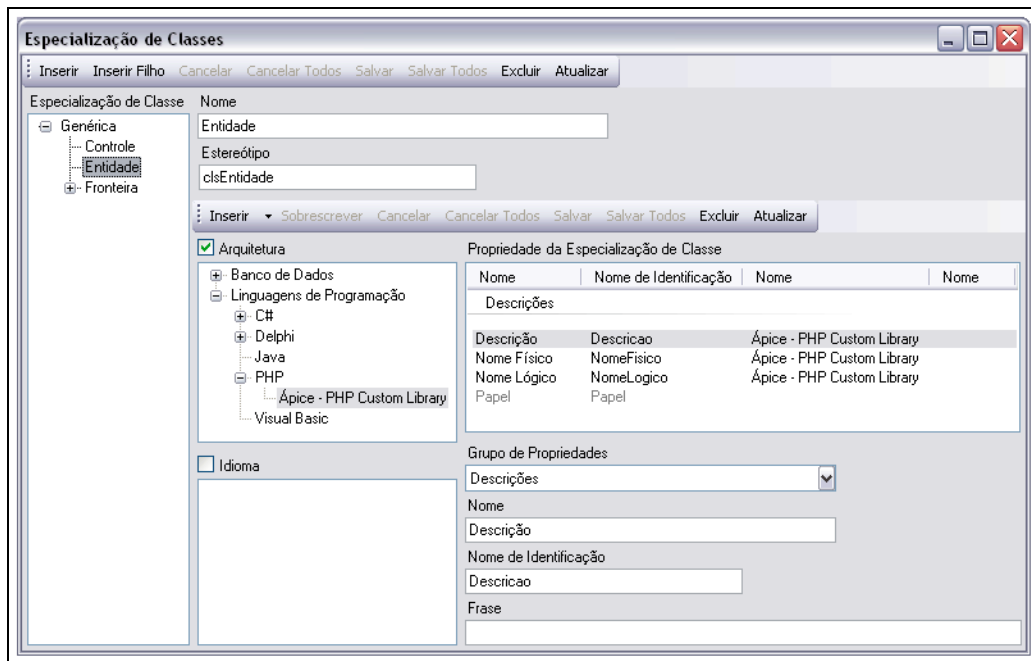
Figura 4: Propriedades de especialização das classes de entidade
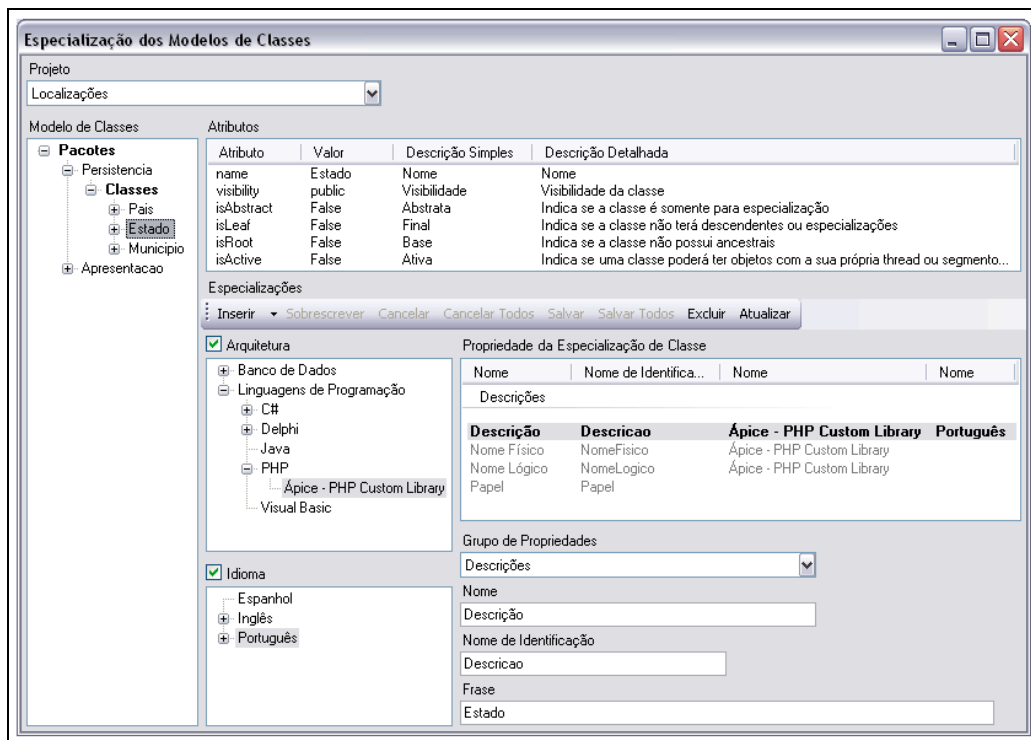
Figura 5: Especialização da classe “Estado”

A sofisticação tecnológica crescente em diversos países e o intercâmbio mundial de tecnologias, resultante tanto do crescimento tecnológico quanto do processo de globalização da economia, implicam em uma necessidade maior de observação de aspectos internacionais no projeto e desenvolvimento de software (ARAÚJO, 2003, p. 45). Sendo assim a especialização por idioma é um recurso indispensável para organizações que realizam a internacionalização de seus softwares.

A Figura 6 apresenta a especialização da propriedade "Descrição" para o idioma inglês. Nota-se que na Figura 5 a mesma propriedade está especializada para o idioma português.
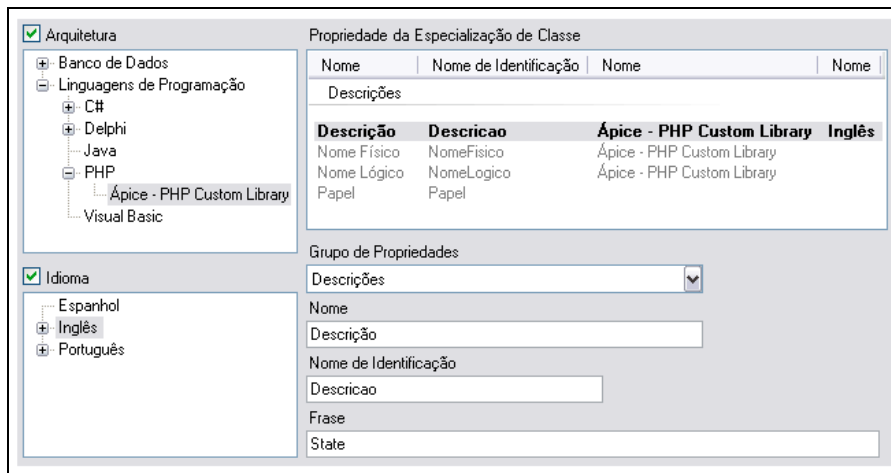
Figura 6: Especialização da propriedade "Descrição" para o idioma inglês

O Quadro 3 apresenta um trecho do código gerado para a classe "Estado" utilizando o idioma português. Nota-se em destaque a utilização da propriedade "Descrição" anteriormente citada.

```
class Estado extends CustomEntity {

    var $attributes = NULL;
    var $views = NULL;
    var $relations = NULL;

    function Estado($owner) {
        $this->CustomEntity($owner);

        $this->setPhysicalName("tb_est");
        $this->setLogicalName("estado");
        $this->setDescription("Estado");
    }

    function estadosPorPais($identificadorPais) {
        return NULL;
    }

    function getAttributes() {
        if ($this->attributes == NULL)
            $this->attributes = new EstadoAtributos($this);

        return $this->attributes;
    }
```

Quadro 3: Trecho do código PCL gerado para a classe "Estado"

Além das classes de entidades utilizadas neste exemplo, o *plugin* de geração de código para a arquitetura PCL também realiza a geração de classes de interface permitindo a criação de um software completo para inclusão, alteração e exclusão de dados, considerando também os relacionamentos entre as classes. Um exemplo de interface gerada utilizando o idioma inglês é apresentada na Figura 7.
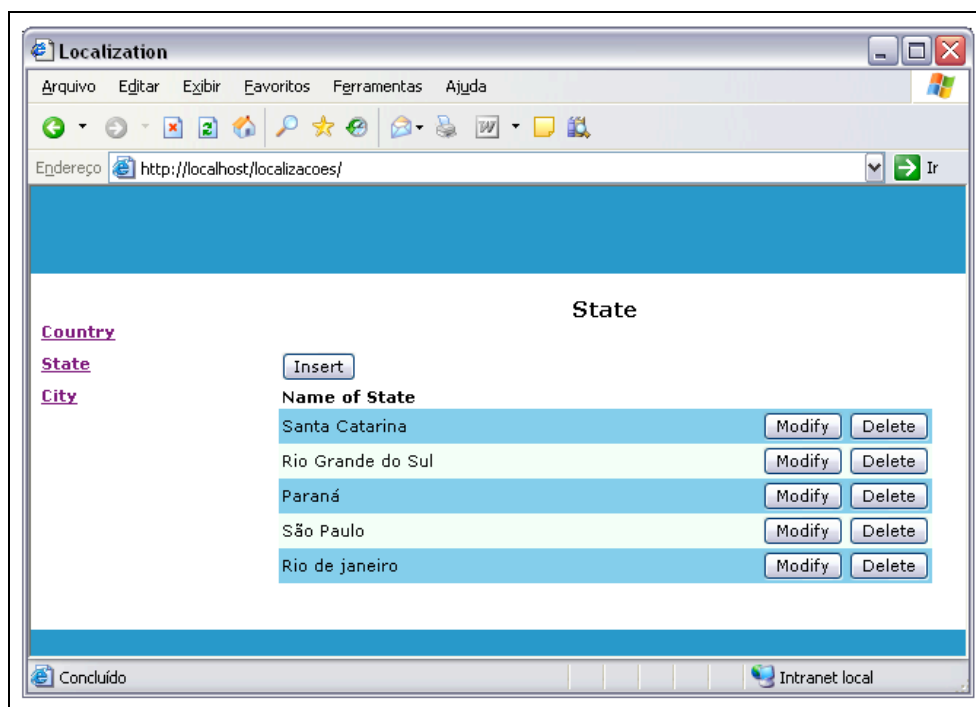
Figura 7: Exemplo de interface PCL gerada para manutenção da classe "Estado"

4 Conclusões

A melhora na qualidade do software desenvolvido e a redução do tempo de desenvolvimento abordados na introdução não são mais vistos pelo mercado como um diferencial, mas sim como requisitos fundamentais para fechamento de negócios. Com base nos problemas citados, o processo de Engenharia da Produção (*Forward Engineering*) através da geração de código tem a função de amenizar estes problemas. A geração de código reduz o tempo do processo de desenvolvimento, pois minimiza a necessidade de codificação manual e aumenta a confiabilidade do código gerado, pois é produzido por uma ferramenta depurada e testada (FISHER, 1990, p.10).

Além do aumento da qualidade e redução do tempo de desenvolvimento, a ferramenta apresentada ajuda a agregar uma melhor padronização ao processo de desenvolvimento, pois a mesma requer o uso da linguagem de documentação e modelagem UML (*Unified Modeling Language*) e de metodologias propostas pela Engenharia de Software. Nota-se também que um processo de desenvolvimento de software bem definido é um grande passo para a obtenção de certificações através dos Sistemas de Garantia da Qualidade.

A estrutura de especialização através da utilização de propriedades genéricas permite a abstração de dados comuns para as diferentes arquiteturas utilizadas, possibilitando desta forma o

reaproveitamento de informações e enriquecimento do repositório de dados da ferramenta. A especialização por arquitetura permite o isolamento de todas as características de uma determinada arquitetura, mantendo desta forma um repositório de dados mais limpo e organizado, facilitando o uso e a manutenção da ferramenta e tornando os softwares gerados mais flexíveis perante as mudanças tecnológicas. A especialização por idioma fornece um recurso fundamental para as organizações na internacionalização de seus softwares, pois através da mesma é possível realizar a geração de código para diferentes idiomas.

A utilização de diagramas de classes especializados permite a construção de *plugins* para geração de código mais inteligentes e com a capacidade de abstrair a utilização de um diagrama de classes de implementação. A abstração citada permite ao analista um foco maior no negócio em questão, poupando também o esforço gasto pela equipe de desenvolvimento na implementação de rotinas básicas.

A ferramenta, suas estruturas e processos foram concebidos através do trabalho de conclusão de curso de Deschamps (2005). Atualmente a ferramenta encontra-se em fase desenvolvimento por uma empresa incubada no Instituto GENE de Blumenau. Pretende-se em curto prazo realizar o lançamento da versão comercial da ferramenta.

Referências

ARAÚJO, Eratóstenes Edson Ramalho de. A internacionalização e a localização de produtos e serviços: a sua importância na indústria de software, **T&C Amazônia**. [S.l.], n. 2, p. 44-48, jun. 2003.

BEZZERA, Eduardo. **Princípios da análise e projeto de sistemas com UML**. Rio de Janeiro: Campus, 2003. 286 p.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML, guia do usuário**. Rio de Janeiro: Campus, 2000. 472 p.

DESCHAMPS, Alexandro. **Ferramenta para geração de código a partir da especialização do diagrama de classes**. 2005. 108 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FISHER, Alan S. **CASE**: utilização de ferramentas para desenvolvimento de software. Rio de Janeiro: Campus, 1990. 264 p.

OBJECT MANAGEMENT GROUP. **OMG XML Metadata Interchange (XMI) Specification**. [S.l.], 2002. Disponível em: < http://www.omg.org/cgi-bin/apps/doc?formal/02-01-01.pdf >. Acesso em: 30 ago. 2004.

SCHMIDT, Roger Anderson. **Ferramenta de auxílio ao processo de desenvolvimento de software integrando tecnologias otimizadoras**. 2001. 115 f. Trabalho de Conclusão de Curso (Bacharel em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

WEHRMEISTER, Marco Aurélio. **Software para geração de código fonte a partir do repositório da ferramenta CASE System Architect**. 2001. 105 f. Trabalho de Conclusão de Curso (Bacharel em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

Estudo de Caso de Aplicação da Métrica de Pontos de Casos de Uso numa Empresa de Software

Viviane Heimberg (Senior Sistemas)
viviane@senior.com.br

Everaldo Artur Grahl (FURB/DSC)
egrahl@furb.br

Resumo: A estimativa de tamanho é uma das métricas de software mais utilizadas, porque a partir dessa dimensão é possível definir o esforço, o prazo e os custos necessários para o desenvolvimento do software. Atualmente uma das métricas que está sendo testada e adotada nas empresas é a PCU (Pontos de Casos de Uso). Neste artigo foi apresentado um estudo de caso de uma empresa de software onde foi utilizada a métrica de Pontos de Casos de uso para aumentar a precisão nas estimativas de tempos de desenvolvimento de três projetos de software Web. O experimento demonstrou a necessidade de se realizar controles mais eficientes na obtenção e análise dos requisitos e em especial dos Casos de Uso, além de ser importante o ajuste de valores de horas para futuras previsões.

Palavras-chave: Métricas; Pontos de casos de uso.

1 Introdução

Na última década, a mudança na tecnologia de comunicação com a popularização da internet influenciou profundamente os processos nas empresas exigindo novas metodologias de análise e desenvolvimento de sistemas de informação. As empresas precisam adequar seus sistemas atuais aos novos volumes de informação, tanto estratégicas quanto operacionais. Segundo Tavares, Carvalho e Castro (2004), este pode ser considerado um desafio para as organizações que desenvolvem softwares, pois precisam construir sistemas úteis e no tempo adequado para atender os níveis de competitividade exigidos pelo mercado. Para alcançar estes objetivos, as empresas estão utilizando metodologias de desenvolvimento onde se torna necessário estimar o tempo das atividades do processo desde a fase de concepção até as fases de produção e testes. Isto tem exigido a adoção de métricas de estimativas de desenvolvimento de sistemas para medir o esforço e o tempo que um software produzirá.

O objetivo deste artigo é relatar um estudo de caso aplicando a métrica Pontos de Casos de Uso (PCU) numa empresa de software verificando o seu comportamento para, posteriormente, propor ajustes para sua melhor utilização. Esta empresa está iniciando a conversão dos seus sistemas para o ambiente Web utilizando uma metodologia de desenvolvimento para sistemas Web baseada no processo unificado de desenvolvimento e ainda não adotou nenhuma métrica para gerenciamento dos seus projetos. A métrica PCU foi escolhida pois permite fazer estimativas no início do projeto com base nos modelos de casos de uso construídos. Esta escolha deu-se devido as suas características intrínsecas serem as mais adequadas na estimativa dos tempos dos projetos orientados a objetos e com foco no ambiente WEB e por ser passível de medição na empresa de software estudada.

Nas seções seguintes é apresentada uma breve descrição sobre a gerência de projetos de software, o processo de desenvolvimento de software, métricas de estimativa de tamanho, a métrica PCU, a coleta de dados na empresa com a análise dos resultados, as conclusões e sugestões para trabalhos futuros.

2 O Processo de desenvolvimento de software

Existem tantas maneiras de desenvolver software quanto existem desenvolvedores. Entretanto, uma equipe de desenvolvimento de software precisa de uma estratégia unificada para desenvolver software. As metodologias de software definem uma maneira comum de encarar o desenvolvimento. Para atender a essa necessidade nasceu o Processo Unificado que é uma estrutura genérica de processo que pode ser customizada adicionando-se ou removendo-se atividades com base nas necessidades específicas e nos recursos disponíveis para um projeto. O Rational Unified Process (RUP) é um exemplo de versão customizada do Processo Unificado.

O Processo Unificado (SCOTT, 2003) faz uso extensivo da Linguagem de Modelagem Unificada - UML. Seus princípios fundamentais são:

- Dirigido por casos de uso: uma seqüência de ações executadas por um ou mais atores (pessoas ou entidades não-humanas fora do sistema) e pelo próprio sistema, que produz um ou mais resultados de valor para um ou mais atores. Os casos de uso não são selecionados isoladamente, os desenvolvedores devem começam pelos casos de uso chave e amadurecendo-os no decorrer do ciclo de vida do sistema.
- Centrado em Arquitetura: onde é descrita a visão do sistema como um todo. É o conjunto de tecnologias e ambientes sobre o qual o software será desenvolvido.
- Iterativo e Incremental: sendo que uma iteração é um mini-projeto que resulta em uma versão do sistema liberada interna ou externamente. Supõe-se que essa versão possua um melhoria incremental sobre a anterior, motivo pelo qual é chamada de incremento.

O Processo Unificado divide-se em quatro fases ilustradas pela figura 1 e listadas a seguir:

- Concepção: o objetivo desta fase é estabelecer a viabilidade do sistema proposto.
- Elaboração: o objetivo desta fase é estabelecer a capacidade para a construção do novo sistema, dadas as restrições financeiras, de cronograma e de outros tipos de restrições com que o desenvolvimento se defronta.
- Construção: o objetivo da fase de construção é construir um sistema capaz de operar bem em ambientes beta de clientes.
- Transição: o objetivo desta fase é a de entregar o sistema completamente funcional aos clientes.

Segundo Scott (2003), cinco fluxos de trabalhos atravessam o conjunto das quatro fases do Processo Unificado. Cada fluxo de trabalho é um conjunto de atividades que vários membros executam. Eles estão descritos a seguir:

- Requisitos**:** visa construir um modelo de casos de uso que captura os requisitos funcionais do sistema que está sendo definido. Este modelo ajuda os interessados no projeto a chegar a um acordo sobre as capacidades do sistema e as condições que ele deve satisfazer.
- Análise: visa construir um modelo de análise, que ajuda os desenvolvedores a refinar e estruturar os requisitos funcionais capturados pelo modelo de casos de uso. Este modelo contém realizações de casos de uso mais apropriadas ao trabalho do projeto e de implementação do que casos de uso.
- Projeto: visa construir um modelo de projeto, o qual descreve as realizações físicas dos casos de uso a partir do modelo de projeto, o qual descreve as realizações físicas dos casos de usos a partir do modelo deste e do conteúdo do modelo de análise. O modelo serve como uma abstração do modelo de implementação.
- Implementação: visa construir um modelo que descreve como os elementos do modelo de projeto,como arquivos de código-fonte, bibliotecas de ligações dinâmicas e componentes executáveis (EJBs, por exemplo), são empacotados em componentes de software.

- Teste: visa construir um modelo de teste, que descreve como os teste de integração e de sistema exercitarão componentes executáveis a partir do modelo de implementação.
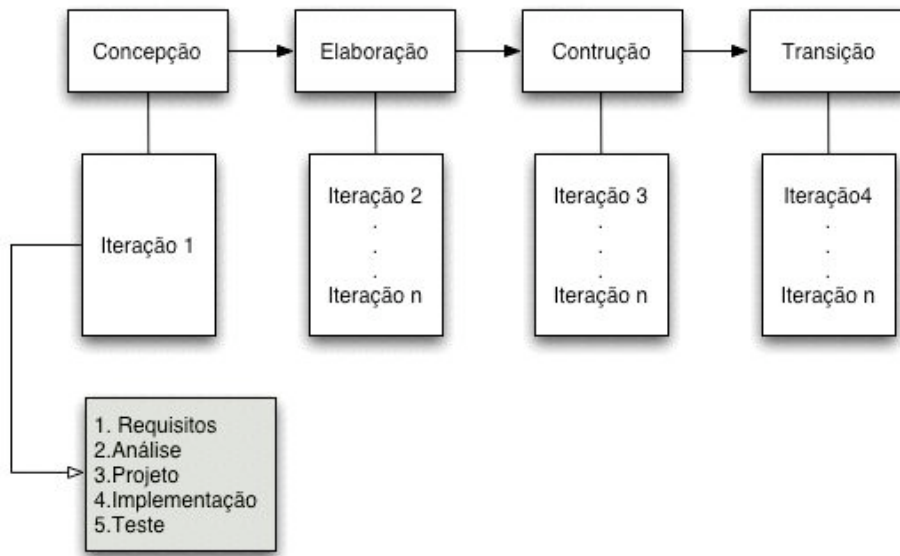
Figura 1: Mecanismo tradicional de iteração do processo unificado

Atualmente, o processo unificado vem sendo estudado, adaptado e adotado gradualmente por muitas empresas de software.

3 Métricas de estimativa de tamanho de software

Para Andrade e Oliveira (2004), métricas são metodologias de mensuração cujos principais objetivos, na área de análise de sistemas, são de estimar o tamanho de um software e auxiliar, como indicador, o gerenciamento dos projetos de desenvolvimento de sistemas. A estimativa de tamanho é uma das métricas mais utilizadas pois o tamanho do software tem impacto direto no esforço de desenvolvimento e na gestão do projeto. É um indicador da quantidade de trabalho a ser executado sendo possível definir esforço, prazo e custos necessários.

Segundo Pressmann (2002), a medição permite aos gerentes planejar, controlar, melhorar e aperfeiçoar o processo de desenvolvimento de software. Medição resulta em mudança cultural. Coletar dados, calcular e analisar métricas são três passos que devem ser implementados para iniciar um programa de métricas. O custo e o esforço necessários para construir software, mesmo em linhas de código produzidas são fáceis de coletar, desde que convenções específicas para a medição sejam estabelecidas antecipadamente. O software orientado a objetos é fundamentalmente diferente do software desenvolvido utilizando métodos convencionais. Por esse motivo as métricas para os sistemas orientados a objetos devem ser ajustadas considerando as características que a distinguem do software convencional. Existem várias métricas que podem ser utilizadas, dentre as quais encontram-se o número de linhas de código e a Análise de Pontos por Função (APF). Esta última muito utilizada nas organizações de software. Maiores informações sobre a Análise de

Pontos de Função podem ser obtidas em Vasquez, Simões e Albert (2003). A métrica Pontos de Casos de Uso foi escolhida por ser uma métrica baseada fortemente na UML e mais especificamente pelos casos de uso e ainda pelo fato da empresa estar iniciando o uso da UML.

3.1 A métrica de Pontos de Casos de Uso (PCU)

Os Pontos de Casos de Uso (PCU) foram criados em 1993 por Gustav Karner da empresa Objectory AB. Esta métrica permite fazer estimativas no início do projeto com base no modelo de casos de uso. A filosofia dos Pontos de Casos de Uso é baseada na definição da Análise de Pontos por Função (APF), na qual a funcionalidade vista pelo usuário é a base para a estimativa do tamanho do software (BELGAMO; FABBRI, 2004).

O processo de contagem dessa métrica consiste nos seguintes passos (MEDEIROS, 2004):

1. Relacionar os atores, classificá-los de acordo com seu nível de complexidade (simples, médio ou complexo) atribuindo respectivamente os pesos 1, 2 ou 3., conforme a tabela 1. Calcule o TPNAA (Total de Pesos não Ajustados dos Atores) somando os produtos da quantidade de atores pelo seu peso.

Complexidade do ator	Descrição	Peso
Simples	Muito poucas entidades de Banco de Dados envolvidas e sem regras de negócio complexas	1
Médio	Poucas entidades de Banco de Dados envolvidas e com algumas regras de negócio complexas	2
Complexo	Regras de negócios complexas e muitas entidades de Bancos de Dados presentes	3

Tabela 1: Classificação de atores (MEDEIROS, 2004)

2. Contar os casos de uso e atribuir o grau de complexidade sendo a complexidade baseada no número de classes e transações. Calcule o TPNAUC (Total de Pesos não ajustados dos casos de usos) somando os produtos da quantidade de casos de usos pelo respectivo peso conforme a tabela 2.

Tipo de Caso de Uso	Descrição	Peso

Simples	Considerar até 3 transações com menos de 5 classes de análise	5
Médio	Considerar de 4 a 7 transações com 5 a 10 classes de análise	10
Complexo	Considerar de 7 transações com pelo menos de 10 classes de análise	15

Tabela 2: Classificação dos Casos de Uso (MEDEIROS, 2004)

3. Calcular PCU´s não ajustados, também chamados de PCUNA, de acordo com a seguinte fórmula:

PCUNA = TPNAA+ TPNAUC

4. Determinar o fator de complexidade técnica. Os fatores de complexidade técnica variam numa escala de 0 a 5, de acordo com o grau de dificuldade do sistema a ser construído. O valor 0 indica que a grau não está presente ou não é influente, 3 influência média e o valor 5 indica influência significativa através de todo o processo. Após determinar o valor dos fatores, multiplicar pelo respectivo peso ilustrado na tabela 3, somar o total e aplicar a seguinte fórmula:

Fator de complexidade técnica (FCT) = 0.6 + (0.01 * Somatório do Fator técnico)

Descrição	Peso
Sistemas Distribuídos	2,0
Desempenho da aplicação	1,0
Eficiência do usuário final (on-line)	1,0
Processamento interno complexo	1,0
Reusabilidade do código em outras aplicações	1,0
Facilidade de instalação	0,5
Usabilidade (facilidade operacional)	0,5
Portabilidade	2,0
Facilidade de manutenção	1,0
Concorrência	1,0
Características especiais de segurança	1,0
Acesso direto para terceiros	1,0
Facilidades especiais de treinamento	1,0

Tabela 3: Fatores de complexidade técnica (MEDEIROS, 2004)

5. Determinar o fator de complexidade ambiental: os fatores de complexidade ambientais indicam a eficiência do projeto e estão relacionados ao nível de experiência dos profissionais. Esses fatores descritos na tabela 4 são determinados através da escala de 0 a 5, onde 0 indica baixa experiência, 3 indica média experiência e 5 indica alta experiência. Após determinar o valor de cada fator, multiplicar pelo peso e somar o total dos valores. Em seguida, aplicar a seguinte fórmula:

6. Fator de complexidade ambiental (FCA) = 1,4 + (-0,03 * Somatório do Fator Ambiental)

7. Calcular os PCU´s ajustados: esse cálculo é realizado com base na multiplicação dos PCU não ajustados, na complexidade técnica e na complexidade ambiental através da seguinte fórmula:

PCUA = PCUNA * Fator de complexidade técnica * Fator de complexidade ambiental

Fator	Descrição	Peso
F1	Familiaridade com o processo de desenvolvimento de software	1,5
F2	Experiência na aplicação	0,5
F3	Experiência com OO, na linguagem e na técnica de desenvolvimento	1,0
F4	Capacidade do líder de análise	0,5
F5	Motivação	1,0
F6	Requisitos estáveis	2,0
F7	Trabalhadores com dedicação parcial	-1,0
F8	Dificuldade da linguagem de programação	-1,0

Tabela 4: Fatores de complexidade ambiental (MEDEIROS, 2004)

8. Calcular a estimativa de horas de programação. Karner, o criador da estimativa, sugere a utilização de 20 pessoas-hora por unidade de PCU. Schneider e Winters sugerem o seguinte refinamento :

X = total de ítens de F1 a F6 com pontuação abaixo de 3
Y = total de ítens de F7 a F8 com pontuação acima de 3
Se X + Y <= 2, usar 20 como unidade de homens/hora
Se X + Y = 3 ou X + Y = 4, usar 28 como unidade de homens/hora
Se X + Y >= 5, deve-se tentar modificar o projeto de forma a baixar o número, pois o risco de insucesso é relativamente alto.

Estimativa de horas = PCUA * pessoas hora por unidade de PCU

3.2 Estudo de caso

A empresa estudada é uma desenvolvedora de software corporativo, fundada há mais de quinze anos, de porte médio, com faturamento anual estimado em torno de R$ 30 milhões e cerca de 300 funcionários. Possui uma área específica de pesquisa para o desenvolvimento de técnicas e de ferramentas de análise e programação, porém ainda não utiliza nenhuma métrica para as estimativas dos tempos de desenvolvimento.

Foram analisados os diagramas de casos de uso dos seguintes três projetos: Sistema de cálculo de Folha de Pagamento (Projeto 1), Sistema Contábil (Projeto 2) e o Sistema de Cartão-Ponto (Projeto 3). Em cada projeto foram realizadas as fases de concepção e a primeira iteração da fase de elaboração. Na fase de concepção somente foram elaborados os diagramas de nível 0 de apenas 1 módulo em cada projeto. Para a primeira iteração da fase de elaboração foram ampliados apenas alguns casos de uso considerados relevantes pelos analistas de cada projeto.

O índice do fator de complexidade ambiental obteve o mesmo valor para os 3 projetos pois os analistas possuíam o mesmo grau de experiência em UML, mesma familiaridade com processo unificado de desenvolvimento, mesma experiência em orientação a objetos, mesmo grau de motivação e mesmo conhecimento do ambiente de desenvolvimento, pois todos receberam um treinamento padronizado antes do início dos projetos. Os requisitos foram considerados estáveis pois trata-se da conversão de 3 sistemas já existentes em ambiente cliente/servidor para o ambiente Web.

O índice do fator técnico do projeto variou apenas nos ítens referentes a complexidade de processamento , concorrência e acesso direto a terceiros. Todos os sistemas executam cálculos complexos e precisos, exigem máxima segurança e possuem mais de cem usuários acessando simultaneamente. Os pesos utilizados para os fatores técnicos e ambientais foram os sugeridos por Medeiros (2004).

As equipes dos respectivos projetos ainda não estão completas e a metodologia de desenvolvimento adaptada do processo unificado ainda está em teste, por isso o total de pessoas-horas por unidade de PCU considerado foi de 20 horas homem para uma primeira análise. As horas estimadas não foram divididas pelo total de membros da equipe. Os resultados desta pesquisa na fase de concepção foram ilustrados na tabela 5 e os da fase de elaboração na tabela 6.

Projetos	Atores	Casos de Uso	FCT	FCA	PCUNA	PCUA	Horas Estimadas
Projeto 1	4	1	1,00	0,81	22	17,93	358,60
Projeto 2	6	2	1,02	0,81	37	30,76	615,16
Projeto 3	7	2	1,03	0,81	39	32,74	654,77

Tabela 5: Estimativas da fase de concepção

Projetos	Atores	Casos de Uso	FCT	FCA	PCUNA	PCUA	Horas Estimadas
Projeto 1	4	5	1,00	0,81	72	58,68	1.173,60
Projeto 2	6	8	1,02	0,81	87	72,32	1,446,46
Projeto 3	7	4	1,03	0,81	64	53,72	1,074,50

Tabela 6: Estimativas da fase de elaboração

Verificou-se junto as equipes de desenvolvimento e aos coordenadores dos projetos que os tempos obtidos pela estimativa estavam muito acima dos obtidos em projetos semelhantes. Os coordenadores dos projetos identificaram que 20 horas/homem por total de tempo de unidade de PCU era um número muito alto e não representava corretamente uma boa média para todos os tipos de tempos por nível de complexidade de casos de uso. A metodologia de desenvolvimento da empresa utiliza uma camada de código que abstrai grande parte da geração de código básica dos seus sistemas, reduzindo a quantidade de horas/homem para realizar o desenvolvimento de casos de uso para aproximadamente 5 horas/homem em casos de uso simples, 9 horas/homem para casos de uso médios e 24 horas/homem para casos de uso complexos.

Foi decidido realizar uma nova estimativa nos três projetos utilizando os casos de uso da fase de elaboração por estar mais completa, desta vez ajustando-se os pesos dos casos de uso para simples=5, médio=10, complexo = 25 e modificar a quantidade de horas/homem para uma média de 10 horas/homem.

Os resultados obtidos foram os seguintes:

Projetos	Atores	Casos de Uso	FCT	FCA	PCUNA	PCUA	Horas Estimadas
Projeto 1	4	5	1,00	0,81	54	43,74	437,40
Projeto 2	6	8	1,02	0,81	76	61,50	615,60
Projeto 3	7	4	1,03	0,81	57	46,17	461,70

Tabela 7: Estimativas da fase de elaboração com os novos pesos ajustados

3.3 Análise dos resultados

O objetivo deste estudo foi verificar a viabilidade de aplicação da métrica PCU nos diagramas elaborados para uma primeira estimativa de horas necessárias para o desenvolvimento dos 3 projetos. O início da coleta destas estimativas se faz necessário para que a empresa comece a formar um histórico de dados através da comparação das horas estimadas com as horas efetivas ao final da fase de construção destes projetos.

Entre as duas fases houve um crescimento considerável de horas estimadas, pois na nova iteração foram melhor esboçados os requisitos nos diagramas de casos de uso. Diagramas de sequência e de classes também foram utilizados melhorando a identificação da complexidade dos casos de uso.

Segundo o coordenador geral das equipes de desenvolvimento, as quantidades de horas calculadas pela métrica PCU com o tempo padrão baseado em 20horas/homens não se aproximaram dos tempos normalmente auferidos pelas equipes de desenvolvimento através da experiência em programação em projetos anteriores. Nos projetos, os tempos calculados ficaram 70% abaixo dos tempos estimados pela métrica. Com o segundo cálculo, com os pesos dos casos de usos e horas/homem ajustados, as estimativas ficaram muito próximas da realidade atual das equipes de desenvolvimento.

Para uma estimativa mais precisa é necessário que todas as iterações da fase de elaboração estejam completas. Porém muitas empresas de software necessitam possuir o conhecimento do esforço ao final da fase de concepção para poderem alocar recursos, negociar prazos, fechar contratos com seus clientes, ou até mesmo para verificar se é viável ou não a continuidade do projeto.

Neste estudo houve dificuldade em identificar na coleta de dados os diagramas produzidos na fase de concepção. É necessário que empresa possua um registro dos diagramas produzidos ao final de cada fase ou mesmo a cada iteração dependendo dos objetivos que espera alcançar. Sugere-se que seja incluída uma atividade na metodologia de desenvolvimento da empresa de software para que o analista identifique e anote em cada diagrama de caso de uso a sua complexidade. O mesmo para os atores envolvidos no sistema.

Não foi possível a continuidade do levantamento de dados porque os projetos foram interrompidos, sendo sua continuidade prevista após esta pesquisa. A empresa estudada espera que após um ano de utilização da métrica PCU em seus projetos seja possível ao final da fase de elaboração um percentual de acerto de 90% em relação ao total de horas estimadas

4 Conclusões

A métrica PCU, assim como outros tipos de métricas, abordam as estimativas teóricas de tempo, e tem a finalidade de direcionar, viabilizar e ajudar controlar as fases do desenvolvimento de software.

O sucesso de um programa de métricas depende de uma série de fatores que variam desde a conscientização dos profissionais envolvidos até a existência de recursos técnicos e humanos necessários para a manutenção e monitoramento do programa. Um programa de medição traz benefícios para o programador a medida em que possibilita dimensionar melhor a carga de trabalho de forma a garantir a qualidade. Com esta informação o desenvolvedor poderá requerer mais recursos bem como dimensionar o valor pecuniário de seu trabalho além de estimar com mais precisão o tempo necessário para o desenvolvimento de suas tarefas. Para o empresário, este terá uma base de informações imprescindível para estimar seus custos e apreçar seus produtos, bem como estabelecer cronogramas com promessas de entrega de produtos em prazos possíveis de serem alcançados.

Em projetos orientados a objetos para que a estimativa de tamanho seja realizada com maior precisão desde o início do projeto pode-se utilizar as métricas Análises de Pontos de Função e Análise de Casos de Uso de forma combinada no momento em que elas são melhores aplicadas no processo de desenvolvimento. Conforme Andrade (2004) a confiança nas estimativas aumenta quando mais de uma forma de estimar é utilizada e, à medida que se obtém mais informações do domínio do software durante o processo de desenvolvimento do projeto, as estimativas serão melhores.

Este artigo foi elaborado para orientar os profissionais e iniciantes na atividade de desenvolvimento de sistemas sobre a importância da boa prática de controles durante a fase de desenvolvimento através da proposta de uma métrica já de domínio da área da ciência da computação.

Como sugestão para trabalhos futuros, a empresa poderá aplicar a métrica nos projetos pilotos nas próximas iterações até que a última iteração da fase de elaboração estiver concluída. Ao término da fase de construção será possível verificar os tempos de desenvolvimento reais em relação aos tempos estimados e propor novos ajustes na métrica PCU.

5 Referências

ANDRADE, Edméia Leonor Pereira; OLIVEIRA, Káthia Marçal. **Uso Combinado de Análise de Pontos de Função e Casos de Uso na Gestão de Estimativa de Tamanho de Projetos de Software Orientado a Objetos.** In III Simpósio Brasileiro de Qualidade de Software. Brasília: 2004

BELGAMO, Anderson; FABBRI, Sandra. **Um Estudo sobre a Influência da Sistematização da Construção de Modelos de Casos de Uso na Contagem dos Pontos de Casos de Uso.** In III Simpósio Brasileiro de Qualidade de Software. Brasília 2004

MEDEIROS,Ernani. **Desenvolvendo Software com UML 2.0**. São Paulo: ed. Makron Books, 2004.

PRESSMAN, Roger S. **Engenharia de software**. São Paulo : ed. McGraw-Hill, 2002.

SCOTT, Kendall. **O Processo Unificado Explicado**.Porto Alegre: ed. Bookman, 2003.

TAVARES, Helena Cristina A. B.; CARVALHO Ana Elizabete S.; CASTRO Jaelson F. B **Medição de Pontos por Função a Partir da Especificação de Requisitos.** Pernambuco : 2004

VAZQUEZ, Carlos Eduardo; SIMÕES, Guilherme Siqueira; ALBERT, Renato Machado. **Análise de pontos de função: medição, estimativas e gerência de projetos.** São Paulo : Érica, 2003.

Ferramenta para Apoio ao Ensino de Introdução à Programação

Karly Schubert Vargas (FURB/BCC)
karly.vargas@gmail.com

Joyce Martins (FURB/DSC)
joyce@furb.br

Resumo. O presente artigo descreve o desenvolvimento de uma ferramenta para apoiar o ensino de introdução à programação. A ferramenta permite que os acadêmicos construam algoritmos em uma linguagem de programação estruturada e em português. Para facilitar o entendimento da lógica de programação, os algoritmos podem ser executados passo a passo com opção para verificar o conteúdo das variáveis declaradas. Ainda, são apresentados os resultados obtidos com o uso da ferramenta durante o 1º semestre de 2005 na disciplina de Introdução à Programação do curso de Ciências da Computação da Universidade Regional de Blumenau (FURB).

Palavras-chave: Algoritmos, Compilador, Introdução à programação, Ensino-aprendizagem.

1 Introdução

O ensino de programação é essencial na grade curricular de um curso de ciência da computação, sendo considerado a base para o entendimento computacional. O ensino de programação acontece em várias disciplinas, específicas ou não sobre o assunto, sendo a primeira delas Introdução à Programação (ou nome similar) oferecida, em geral, no 1º semestre do curso. Essa disciplina normalmente aborda os princípios da lógica de programação, objetivando que o aluno desenvolva a capacidade de análise e resolução de problemas descrevendo-os através de algoritmos. Segundo as diretrizes curriculares do MEC (MINISTÉRIO DA EDUCAÇÃO, 1999, p.6), "O desenvolvimento de algoritmos, juntamente com o estudo de estruturas de dados deve receber especial atenção na abordagem do tema programação."

Este processo de ensino apresenta dois grandes desafios. O primeiro é despertar a criatividade necessária para o desenvolvimento de soluções computacionais para os problemas. O segundo é representar a solução usando lógica de programação.

A forma usada para representar um algoritmo é variada, podendo destacar-se fluxograma e Portugol. Conforme Saliba (1994), o fluxograma faz uso de símbolos geométricos que representam as estruturas de um programa. Estes símbolos são conectados por arestas dirigidas que fornecem a seqüência de execução. Já o Portugol é uma linguagem que permite representar um algoritmo fazendo uso da língua portuguesa. Através de estruturas básicas (seqüência, seleção ou repetição), é possível construir programas usando uma sintaxe que se aproxima das linguagens de programação usuais. No entanto, independente da forma escolhida, em geral, a descrição do algoritmo é feita no papel.

Aliados à dificuldade para representar o algoritmo e ao uso do papel, surgem questionamentos do tipo: Por que a solução proposta não é adequada? Qual o "caminho" que a solução proposta está seguindo? Estes questionamentos estão ligados ao fato de que a lógica de programação apresenta um grau de abstração inicial grande, pois muitas vezes o aluno não consegue visualizar o que aconteceria se a solução por ele proposta fosse executada em um computador.

Em função disso, várias ferramentas foram desenvolvidas (ALMEIDA et al., 2002; SANTIAGO; DAZZI, 2004; TAGLIARI, 1996; USP, 2004) para auxiliar no processo de ensino-

aprendizado da lógica de programação. Algumas utilizam representação gráfica (fluxograma), outras usam representação textual (Portugol). No entanto, a sintaxe dessas representações difere de ferramenta para ferramenta e isso, segundo Medeiros e Dazzi (2002 apud SANTIAGO; DAZZI, 2004), "torna difícil e desagradável o processo de aprendizagem que deveria ser na medida do possível fácil e prazeroso". Portanto, é recomendado o uso de ferramentas que sejam adequadas para a realidade de cada curso de computação.

Tendo em vista os aspectos apresentados, este artigo descreve uma ferramenta para dar apoio ao ensino de Introdução à Programação, disciplina do curso de Ciências da Computação da Universidade Regional de Blumenau (FURB). A ferramenta proposta permite o desenvolvimento de algoritmos em uma linguagem de programação estruturada e em português (representação textual).

A seguir são apresentadas a importância do computador no ensino e suas classificações, a linguagem de programação especificada e a ferramenta desenvolvida. Também são apresentados os resultados alcançados a partir de uma experiência inicial. Considerações sobre a ferramenta proposta são relatadas na última seção.

2 Ensino com o computador

Segundo Galvis-Panqueva (1997), com a massificação da informática em empresas, lares e escolas, proliferaram os softwares voltados para a educação. Usa-se o computador na educação visando auxiliar o processo de ensino ou desenvolver melhores meios de aprendizado. Baranauskas et al. (1999) define três classes de sistemas computacionais em educação: ensino assistido por computador, aprendizado socialmente distribuído e ambiente interativo de aprendizagem.

2.1 Ensino assistido por computador

No ensino assistido por computador, o computador é visto como uma ferramenta para armazenamento, representação e transmissão de informação. Essas informações são divididas em módulos, que mostram o assunto de maneira gradual e seqüencial. E, geralmente após a apresentação de um módulo, o usuário é submetido a perguntas que devem ser respondidas de acordo com o material apresentado.

Este tipo de sistema surgiu na década de 60, porém, devido a tecnologia pouco desenvolvida, era rígido e não despertava o interesse do aluno, sendo apenas uma nova forma de apresentar conteúdo, um computador ao invés de um livro. Na década de 70, foi desenvolvido o *Intelligent Computer Assisted Learning* (ICAI), no qual a seqüência da apresentação passa a ser personalizada de acordo com o conhecimento demonstrado pelo usuário, tornando o sistema mais atrativo. Estes sistemas foram evoluindo à medida que novas tecnologias e técnicas de inteligência artificial foram surgindo e hoje são chamados de *Intelligent Tutoring Systems* (ITS) ou Tutores Inteligentes (TI) (BARANAUSKAS et al., 1999).

2.2 Aprendizado socialmente distribuído

O aprendizado socialmente distribuído surge com a Internet e a globalização da informação. Este tipo de sistema permite que o conhecimento possa estar numa área comum onde todos possam buscá-lo e ao mesmo tempo acrescentar novos conhecimentos. O potencial deste tipo de sistema é que ele é útil não só ao estudante, mas também ajuda na formação dos professores. A Internet é o meio utilizado para ligar o conhecimento de diversas pessoas e estes conteúdos estão disponível na *World Wide Web* (WWW).

2.3 Ambientes interativos de aprendizagem

Nos ambientes interativos de aprendizado, "o aprendizado é entendido como a construção individual do conhecimento a partir de atividades de exploração, investigação e descoberta." (BARANAUSKAS et al., 1999, p. 50). Os ambientes interativos de aprendizado são baseados em quatro princípios: o estudante deve construir seu conhecimento; o controle do sistema é feito, de forma mais significativa, pelo estudante; o sistema é individualizado para cada estudante; e o *feedback* é gerado em função da interação do estudante com o ambiente. Algumas categorias desta classe de sistemas são: sistemas de modelagem e simulação e ambientes de programação.

Em um sistema de modelagem e simulação típico, o usuário constrói um modelo de um fenômeno/objeto que deseja estudar, utilizando primitivas específicas para representação do modelo, presentes no ambiente computacional. Feito o modelo, o sistema o executa (simula) e apresenta os resultados da simulação. O usuário observa a simulação e pode analisar os resultados obtidos, comparando o modelo construído com o sistema real.

Já os ambientes de programação possuem um grande destaque como ferramenta educacional, uma vez que proporcionam a oportunidade de não só verificar o aprendizado do aluno, mas também de observar como o aluno chegou a determinado resultado. Neles o aluno descreve a solução de um problema, utilizando uma linguagem de programação.

3 Especificação da linguagem de programação

A linguagem utilizada para descrever os algoritmos na ferramenta é seqüencial e imperativa. Foi definida de acordo com a sintaxe usada na disciplina de Introdução à Programação do curso de Ciências da Computação da FURB. Possui as seguintes características:

a) os identificadores iniciam com uma letra e podem conter uma seqüência de letras, dígitos ou o caractere *underline* (_), sendo que nos identificadores as letras maiúsculas e minúsculas são diferenciadas (*case sensitive*);
b) as palavras reservadas são casos especiais de identificadores, porém não são *case sensitive;*
c) as constantes inteiras possuem apenas dígitos e as constantes reais são compostas por dígito(s), o caractere ponto (**.**) e outro(s) dígito(s);
d) as constantes do tipo cadeia são delimitadas pelo caractere ' e podem conter quaisquer caracteres, com exceção de quebra de linha;
e) os comentários de bloco são quaisquer seqüências de caracteres que estiverem entre chaves;
f) os comandos são em língua portuguesa e têm estrutura semelhante ao Pascal com declaração de tipos (somente matriz unidimensional), declaração de variáveis, comando de entrada via teclado, comando de saída em vídeo, comando de atribuição, comandos de seleção (se-então, escolha-caso) e comandos de repetição (enquanto-faça, repita-até, para-faça).

A linguagem é estruturada e não possui suporte a procedimentos, portanto, possui somente um bloco, que inicia com a palavra **ALGORITMO** seguida de um identificador (id) e de ponto e vírgula, e termina com a palavra **FIM** seguida de ponto. A estrutura sintática dos comandos da linguagem é apresentada no Quadro 1, com a notação BNF estendida[1].

[1] Na notação BNF estendida tem-se que: {R}* indica ocorrência de R zero ou mais vezes; {R}+ indica ocorrência de R uma ou mais vezes; {R}? indica ocorrência de R zero ou uma vez.

```
<algoritmo>::=      ALGORITMO id ;
                         {<declaração_tipo>}*
                         {VAR {<declaração_variável>}+}?
                    INÍCIO {<comando>}+ FIM .

<declaração_tipo>::= TIPO id {, id }* = MATRIZ [ cte_inteira .. cte_inteira ] <tipo_primitivo> ;
<declaração_variável>::= id {, id }* : <tipo> ;
<tipo>::= id | <tipo_primitivo>
<tipo_primitivo>::= INTEIRO | REAL | LÓGICO | CARACTERE | CADEIA

<comando>::= <atribuição> | <entrada> | <saída> | <repetição> | <seleção>
<atribuição>::= <variável>:= <expressão> ;
<entrada>::= LEIA ( <variável> {, <variável>}* ) ;
<saída>::= ESCREVA ( <dado> {, <dado> }* ) ;
<dado>::= <variável> | <número> | cte_cadeia
<variável>::= id {[ <expressão> ]}?
<número>::= cte_inteira | cte_real
<repetição>::= ENQUANTO <expressão> FAÇA {<comando>}+ FIMENQUANTO ;
             | REPITA {<comando>}+ ATÉ <expressão> ;
             | PARA id DE <expressão> ATÉ <expressão> FAÇA {<comando>}+ FIMPARA ;
<seleção>::=  SE <expressão> ENTÃO {<comando>}+ {SENÃO {<comando>}+}? FIMSE ;
           |  ESCOLHA id
                         {CASO <opção> : {<comando>}+}+
                         {SENÃO {<comando>}+}?
              FIMESCOLHA
<opção> → cte_inteira | cte_cadeia
```

Quadro 1: Estrutura sintática

No quadro 1, <expressão> representa qualquer expressão aritmética, lógica ou relacional.

4 Desenvolvimento da ferramenta

A ferramenta foi especificada com orientação a objetos, usando *Unified Modeling Language* (UML), através de casos de uso e diagrama de classes, e atende aos seguintes requisitos:

a) possuir um editor para digitação de algoritmos, com as características de um editor de texto convencional, com comandos para edição de textos (recortar, copiar, colar, selecionar tudo) e manipulação de arquivos (abrir, salvar, salvar como), entre outros;

b) compilar os algoritmos escritos em uma linguagem de programação estruturada e em português, gerando um código intermediário e informando os possíveis erros detectados;

c) executar[2] algoritmos até o fim ou passo a passo, sendo que a execução será feita com base no código intermediário gerado. No decorrer da execução passo a passo deve ser possível acompanhar o valor de cada variável declarada no algoritmo.

Três foram os casos de uso identificados: digitar, compilar e executar o algoritmo. O primeiro caso de uso refere-se à digitação do algoritmo: o usuário possui um editor para a digitar um novo algoritmo com a opção de salvar o mesmo, além da opção para abrir um algoritmo já existente. O quadro 2 apresenta o caso de uso da digitação do algoritmo.

[2] Os algoritmos serão executados por um interpretador. Segundo Price e Toscani (1995, p.5), interpretadores "são processadores que aceitam como entrada o código intermediário de um programa anteriormente traduzido e produzem o efeito de execução do algoritmo original".

UC.1. Digitar o algoritmo: o aluno deve digitar o algoritmo, seguindo a sintaxe da linguagem (estruturada e em português) ensinada pelo professor da disciplina de Introdução à Programação.	
Pré-condições	Não possui.
Fluxo principal	1. Selecionar a opção **Novo.** 2. Digitar o algoritmo. 3. Selecionar a opção **Salvar**, informando um nome para o algoritmo.
Fluxo alternativo	1. Existe a opção de abrir um algoritmo já existente. Para isso, deve ser selecionada a opção **Abrir** e informado o nome do arquivo que contém o algoritmo desejado.
Fluxo de exceção	Não há.
Pós-condições	Será habilitada a opção **Compilar**.
Requisitos atendidos	1. Possuir um editor para digitação de algoritmos.

Quadro 2: Caso de uso - digitar algoritmo

Após a digitação do algoritmo, é possível a compilação do mesmo. O processo de compilação do algoritmo verifica se não existem erros léxicos, sintáticos ou semânticos no algoritmo digitado e, em caso negativo, é gerado um código intermediário. O quadro 3 mostra o caso de uso do processo de compilação.

UC.2. Compilar o algoritmo: verificar se o algoritmo digitado não possui erros léxicos, sintáticos ou semânticos, gerando código intermediário.	
Pré-condições	Deve haver um algoritmo digitado no editor da ferramenta.
Fluxo principal	1. Selecionar a opção **Compilar.** 2. A ferramenta faz a análise do algoritmo informando se foram encontrados erros léxicos, sintáticos ou semânticos. 3. É gerado um código intermediário que será utilizado para a execução.
Fluxo alternativo	Não há.
Fluxo de exceção	1. Se não houver um algoritmo digitado, ocorre um erro sintático (início do algoritmo esperado). 2. A geração do código intermediário e a habilitação da opção para executar o algoritmo somente serão realizadas se não houver nenhum erro léxico, sintático ou semântico.
Pós-condições	Será habilitada a opção **Executar**.
Requisitos atendidos	1. Compilar algoritmos escritos em uma linguagem de programação estruturada e em português. 2. Informar os possíveis erros detectados. 3. Gerar código intermediário que possa ser executado.

Quadro 3: Caso de uso - compilar algoritmo

Com base no código intermediário gerado, o processo de execução do algoritmo executa o algoritmo compilado. Para realizar este processo o usuário possui duas opções: executar o algoritmo até o fim ou executar o algoritmo passo a passo, visualizando o valor das variáveis declaradas. Esse caso de uso é apresentado no quadro 4.

Aluno — Executar o algoritmo

UC.3. Executar o algoritmo: executar o algoritmo compilado com base no código intermediário gerado.	
Pré-condições	O algoritmo deve ter sido compilado e não possuir erros léxicos, sintáticos ou semânticos.
Fluxo principal	1. Selecionar a opção **Executar**. 2. A ferramenta abre outra janela. Nela o aluno visualiza o código do algoritmo, as variáveis do algoritmo juntamente com seu valor atual e uma área para entrada e saída de dados. 3. Selecionar a forma de execução: Se optar pela execução até o fim, o algoritmo será executado, parando somente quando necessitar de uma entrada via teclado. Se optar pela execução passo a passo, o algoritmo será executado linha por linha, sendo que após a execução de cada linha, o valor das variáveis é atualizado. 4. Ao final da execução é apresentada a mensagem "FIM DE EXECUÇÃO"
Fluxo alternativo	Não há.
Fluxo de exceção	1. Caso ocorra algum erro durante a execução, o erro será informado ao aluno e a execução cancelada.
Pós-condições	O aluno saberá se seu algoritmo funciona adequadamente.
Requisitos atendidos	1. Permitir que o código intermediário gerado possa ser executado. 2. Permitir que a execução do algoritmo seja feita passo a passo, isto é, linha por linha. 3. Ter uma área para acompanhar o valor das variáveis durante a execução passo a passo do algoritmo.

Quadro 4: Caso de uso - executar algoritmo

A ferramenta possui duas classes principais:

a) **compilador** (`TCompilador`): verifica se o algoritmo digitado não possui erros, gerando código intermediário. Possui como atributos os analisadores léxico[3] (`TLexico`), sintático[4] (`TSintatico`) e semântico[5] (`TSemantico`), além de um atributo para armazenar uma possível mensagem de erro (`msgDeErro`) e outro que indica o *token* que contém o erro (`tokenErrado`). O analisador sintático controla todo o processo de compilação, chamando o analisador léxico quando um *token* é necessário e o analisador semântico para a verificação da semântica estática e geração de código intermediário.

[3] O analisador léxico fragmenta o código fonte em *tokens* (símbolos básicos da linguagem), classificando-os em categorias, tais como palavras reservadas, identificadores, constantes e símbolos especiais.

[4] O analisador sintático tem por função verificar se os *tokens*, recebidos da análise léxica, constituem estruturas sintáticas de acordo a gramática especificada para a linguagem.

[5] O analisador semântico determina o significado de cada construção sintática, traduzindo o programa escrito na linguagem fonte para uma representação intermediária.

Caso ocorra algum erro durante a compilação de um algoritmo, é gerada uma exceção (`ELexicalError`, `ESyntaticError` ou `EsemanticError`) para que o compilador possa tratar erro e apresentar um diagnóstico adequado, informando ao usuário qual o erro encontrado no algoritmo e como corrigi-lo;

b) **interpretador** (`TInterpretador`): executa o algoritmo compilado com base no código intermediário gerado. A classe `TInterpretador` possui o atributo `memória` (um objeto da classe `TMemoria`), cuja função é armazenar os valores na memória durante a execução do algoritmo, e o atributo `ponteiro`, que indica a próxima instrução a ser executada. Possui métodos para: chamar a próxima instrução a ser executada, reiniciar a execução de um algoritmo e verificar compatibilidade de tipos em tempo de execução. Também foram especificados 41 métodos que são as instruções da linguagem intermediária.

Para implementação do protótipo foram utilizados o ambiente de desenvolvimento Borland Delphi 7 e uma ferramenta para geração de analisadores léxicos e sintáticos (GESSER, 2003).

5 Operacionalidade da ferramenta

Quando a ferramenta é executada, a tela apresentada na figura 1 é exibida. Nela o aluno deve informar seu nome e selecionar o botão **Entrar** para fazer uso da ferramenta.

Figura 1: Tela de abertura da ferramenta

A figura 2 apresenta o ambiente para desenvolvimento de algoritmos em Portugol. Ele é composto por um editor de textos com opções para edição de textos e manipulação de arquivos, por um compilador (opção **Compilar**) e por um interpretador (opção **Executar**). O aluno pode acessar as funções do ambiente fazendo uso dos botões e das teclas de atalho, como também dos menus. Na parte inferior da tela é apresentado o *status* atual do editor, informando a linha atual do *cursor*, o nome do algoritmo aberto e se o mesmo foi ou não modificado.

No editor de textos, as palavras reservadas são apresentadas em negrito, as constantes numéricas na cor vermelha, os comentários em itálico e na cor verde, e as constantes do tipo cadeia na cor azul. O uso de cores visa facilitar a identificação dos comandos por parte dos alunos.

```
algoritmo teste;
var a:inteiro; b:cadeia;
inicio
  {Entrada de dados}
  leia(b);
  {Comando de seleção}
  se b = 'dia' então
     escreva('bom dia');
  senão
     escreva('boa tarde');
  fimse;
  leia(a);
  se a >= 18 então
      escreva('maior de idade, com ',a);
  senão
      escreva ('sua idade é: ',a);
  fimse;
fim.
```

```
Algoritmo compilado com sucesso.
```
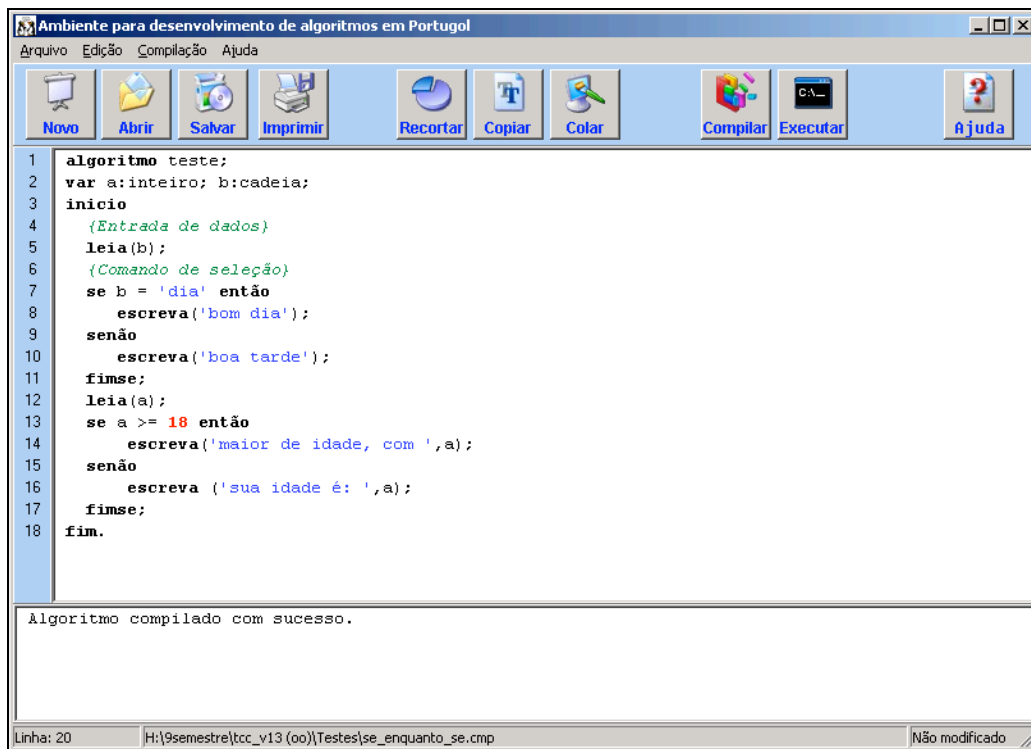
Figura 2: Tela da ferramenta com algoritmo digitado

Quando a opção **Compilar** é selecionada, as mensagens de erro ou a informação de que o algoritmo foi compilado com sucesso são apresentadas na parte inferior da tela.

Quando a opção **Executar** é selecionada, uma nova tela é aberta. A figura 3 apresenta esta tela. Ela é composta por quatro áreas, sendo elas:

a) *console*: é a área onde o aluno irá fornecer os dados quando um comando de entrada for executado e onde serão apresentados os dados quando um comando de saída for executado;

b) variáveis: contêm o nome das variáveis declaradas e seus respectivos valores. Caso a opção passo a passo seja utilizada, a cada linha executada, o valor das variáveis é atualizado;

c) algoritmo: mostra o algoritmo que será executado. Caso a opção passo a passo seja utilizada, a linha que está sendo executada no momento é destacada;

d) botões: são as opções de execução. O primeiro é para habilitar a execução passo a passo. Nesse caso, para que a próxima linha seja executada, este botão deve ser selecionado. Se não desejar executar o algoritmo passo a passo, o botão “Executar até FIM” deve ser selecionado. Caso deseje começar a execução novamente, o botão “Executar de novo” deve ser selecionado.
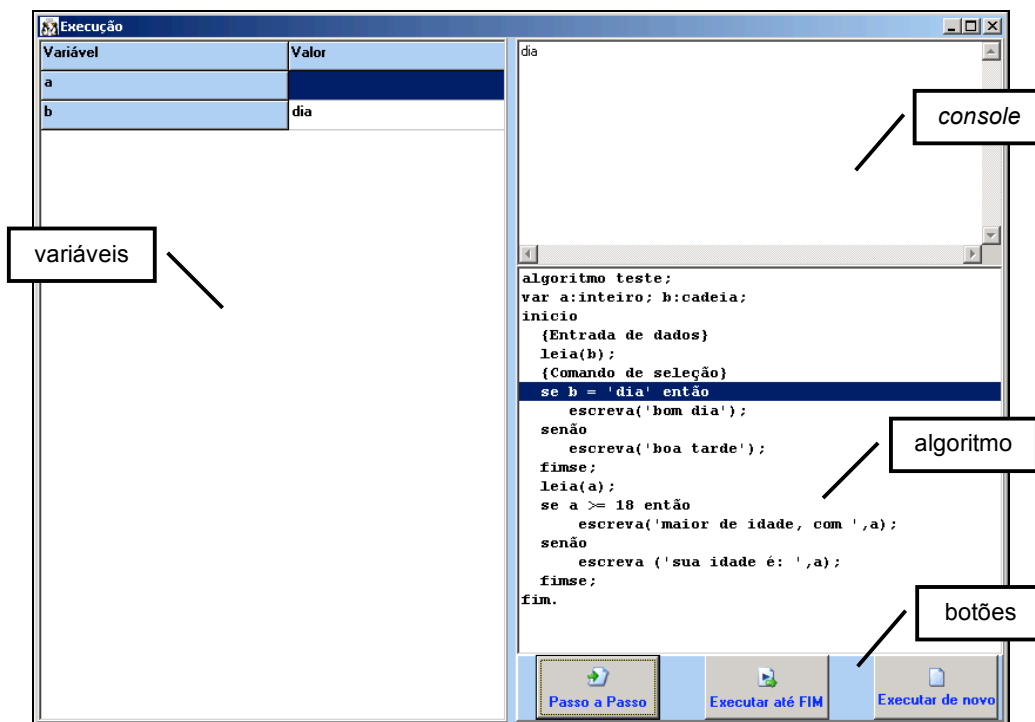
Figura 3: Tela de execução

Para fechar a tela basta selecionar o botão com X no lado direito superior.

A ferramenta também possui uma ajuda sobre Portugol. Ela pode ser utilizada de três maneiras. A primeira delas é selecionar o botão Ajuda da tela principal (figura 2). A segunda maneira é apertar o botão F1. Em ambos casos, será aberta a tela com as opções de ajuda (figura 4). Caso o aluno deseje informações sobre um comando em particular, basta selecionar no editor do ambiente a palavra reservada referente ao comando e pressionar o botão F1. Com isto a ajuda será aberta já no tópico referente ao comando selecionado. A figura 5 apresenta o tópico de ajuda do comando de entrada de dados.
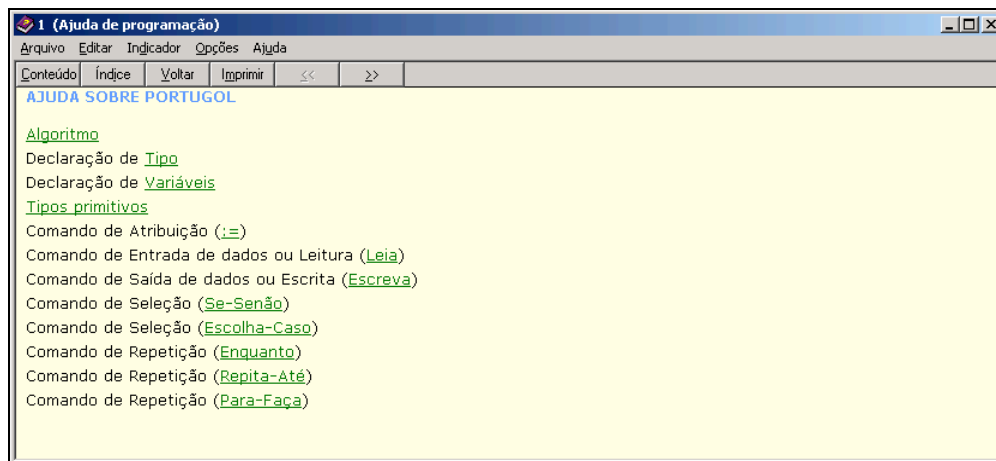
Figura 4: Tela principal da ajuda

5 (Ajuda de programação)

Arquivo Editar Indicador Opções Ajuda

Conteúdo Índice Voltar Imprimir << >>

COMANDO DE ENTRADA DE DADOS OU LEITURA (LEIA)

Como escrever?

```
LEIA ( <lista de variáveis> ) ;
```

onde:

<lista de variáveis> : contém a relação dos nomes (ou **identificador**es) das variáveis nas quais serão armazenados os valores digitados. As variáveis só podem armazenar valores do tipo declarado, conforme a declaração de variáveis.

EXEMPLO: quando da execução do algoritmo abaixo, deverão ser digitados primeiramente valores do tipo **CADEIA** que serão armazenados nas variáveis *nome01* e *nome02*. E quando da execução do segundo comando **LEIA**, deverão ser digitados valores do tipo **REAL** para *salário01* e *salário02*.

```
ALGORITMO com_entrada_de_dados ;
VAR
    salário01 , salário02 , média : REAL ;
    nome01, nome02 : CADEIA ;
INÍCIO
    ESCREVA ( 'digite o nome de dois funcionários:' );
    LEIA ( nome01 , nome02 );
    ESCREVA ( 'digite o salário de cada funcionário:' );
    LEIA ( salário01 , salário02 );
    média := ( salário01 + salário02 ) / 2 ;
    ESCREVA ( 'o salário de ', nome01, 'é: ', salário01 );
    ESCREVA ( 'o salário de ', nome02, 'é: ', salário02 );
    ESCREVA ( 'a média aritmética dos salários é:', média );
FIM .
```

Figura 5: Tela de ajuda do comando leia

Nas figuras anteriores, as palavras sublinhadas em verde fornecem explicações adicionais.

6 Resultados

A ferramenta foi utilizada durante o 1º semestre de 2005 na disciplina de Introdução à Programação do curso de Ciências da Computação da FURB, nos períodos matutino e noturno.

Para avaliar a ferramenta foram aplicados dois questionários aos alunos, um no início e outro no final do semestre. Foi possível verificar que os alunos acharam a interface da ferramenta adequada e, conseqüentemente, de uso fácil e prático. Destaca-se o fato da ferramenta ter apresentado respostas adequadas às ações feitas, tanto em relação ao resultado das ações no ambiente, quanto em relação às mensagens de erros encontrados no algoritmo. 100% dos alunos responderam que o uso de um ambiente para desenvolvimento de algoritmos auxilia no aprendizado da disciplina, sendo que mais de 90% achou que a ferramenta apresentada ajudou no aprendizado. Ressalta-se ainda que 70% dos alunos utilizou a ferramenta fora da sala de aula, o que demonstra interesse por buscar conhecimento extra classe. Foi possível verificar ainda, através de um *log* implementado, quando o aluno utilizou a ferramenta, quais os algoritmos digitados e os erros encontrados na compilação. É importante salientar que os alunos sabiam da existência e do conteúdo destes *logs*.

O professor da disciplina também fez uma avaliação da ferramenta. Quanto às mudanças no ensino de algoritmos, em virtude do uso da ferramenta, o professor destacou como característica importante a possibilidade de apresentar ao aluno a execução passo a passo do algoritmo, permitindo analisar de forma bastante clara as alterações dos conteúdos das variáveis e dos

elementos das matrizes na "memória". Ressaltou também que os alunos elogiaram a ferramenta, comentando que facilitou o entendimento e a visualização da execução.

7 Considerações finais

O ensino de programação é um dos grandes desafios na área de ensino de computação, visto que a dificuldade encontrada pelos alunos é bastante elevada. É essencial o desenvolvimento de ferramentas que busquem despertar o interesse do aluno, assim como facilitar o entendimento da lógica de programação. Por este motivo, foi desenvolvida a ferramenta para auxiliar no ensino de introdução à programação.

A ferramenta é um ambiente interativo de aprendizagem. É importante que este tipo de ferramenta seja desenvolvido de forma personalizada para cada curso, uma vez que as formas usadas para representar algoritmos são diversificadas. Assim, a ferramenta foi desenvolvida com base na sintaxe utilizada para construção de algoritmos pelo professor da disciplina de Introdução à Programação do curso de Ciências da Computação da FURB.

Para facilitar o entendimento da lógica de programação, os algoritmos podem ser executados passo a passo com opção para verificar o conteúdo das variáveis declaradas. Além disso, foi implementada detecção de erros para fornecer um diagnóstico do(s) problema(s) encontrado(s) no algoritmo, indicando o local onde está o erro bem como uma possível solução para o mesmo. Foram tratados erros como: símbolos léxicos incorretos, construções sintáticas inválidas, variáveis declaradas e não utilizadas, variáveis não inicializadas, entre outros. Com base na avaliação da ferramenta feita pelos alunos e pelo professor da disciplina, pode-se afirmar que os objetivos estabelecidos foram atendidos.

Com o estudo feito conclui-se que é importante o uso de ferramentas adequadas ao ensino de determinado assunto, pois permitem experiências que não são possíveis em sala de aula. Especificamente, o ensino de lógica de programação deve ser independente de se utilizar determinada linguagem. Deve ser dada ênfase na construção de algoritmos, etapa tão necessária para o desenvolvimento passo a passo da solução de um problema. Nesse sentido, o uso da ferramenta desenvolvida permite que o aluno visualize o que acontece quando a solução proposta por ele é executada em um computador, o que facilita no entendimento da lógica do algoritmo.

Referências

ALMEIDA, Eliana S. et al. AMBAP: um ambiente de apoio ao aprendizado de programação. In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO, 10., 2002 , Florianópolis. **Anais...** Florianópolis: SBC, 2002. 1 CD-ROM.

BARANAUSKAS, Maria C. C. et al. Uma taxonomia para ambientes de aprendizado baseados no computador. In: VALENTE, José A. (Org.). **O computador na sociedade do conhecimento**. São Paulo: USP; Estação Palavra, 1999. p. 45-68. Disponível em: <http://www.inf.ufsc.br/~edla/mec/>. Acesso em: 23 ago. 2004.

GALVIS-PANQUEVA, Álvaro H. Software educacional multimídia: aspectos críticos no seu ciclo de vida. **Revista Brasileira de Informática na Educação**, Florianópolis, n. 1, set. 1997. Disponível em: <http://www.sbc.org.br/index.php?language=1&subject=100&content=magazine &option=content&id=24>. Acesso em: 27 ago. 2004.

GESSER, Carlos E. **GALS**: gerador de analisadores léxicos e sintáticos. 2003. 150 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis.

MINISTÉRIO DA EDUCAÇÃO. **Diretrizes curriculares de cursos da área de computação e informática**. Brasília, 1999. Disponível em: <http://www.mec.gov.br/sesu/ftp/curdiretriz/computacao/co_diretriz.rtf>. Acesso em: 1 jun. 2005.

PRICE, Ana M. A.; TOSCANI, Simão S. **Implementação de linguagens de programação**: compiladores. 2. ed. Porto Alegre: Sagra Luzzatto, 2001.

SALIBA, Walter L. C. **Técnicas de programação**: uma abordagem estruturada. São Paulo: Makron, 1994.

SANTIAGO, Rafael; DAZZI, Rudimar L. S. Ferramenta de apoio ao ensino de algoritmos. In: SEMINÁRIO DE COMPUTAÇÃO, 13., 2004, Blumenau. **Anais eletrônicos...** Blumenau: FURB, 2004. Disponível em: <http://www.inf.furb.br/seminco/2004/artigos/96-vf.pdf>. Acesso em: 27 set. 2004.

TAGLIARI, Alessandra. **Protótipo de um software para o auxílio ao aprendizado de algoritmos**. 1996. 79 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

USP. **ASA**: ambiente de simulação e animação de algoritmos. São Paulo, 2004. Disponível em: < http://www.edsoft.futuro.usp.br/asp-bin/softprop.asp?SE=0&ID=602&SS=construtor>. Acesso em: 24 ago. 2004.

Estacionamento de um Veículo de Forma Autônoma Simulado em um Ambiente Tridimensional Realístico

Milton Roberto Heinen (UNISINOS/PIPCA)
mheinen@turing.unisinos.br

Fernando Santos Osório (UNISINOS/PIPCA)
fosorio@unisinos.br

Farlei José Heinen (UNISINOS/ENGCOMP)
farleih@gmail.com

Resumo. Este trabalho tem por objetivo apresentar um sistema de simulação do controle inteligente de veículos autônomos. O sistema desenvolvido é responsável pela automatização da tarefa de condução de um veículo, onde se buscou obter um sistema de controle robusto capaz de simular de modo realista a ação de estacionar um veículo em uma vaga paralela. O sistema permite controlar o carro através da leitura de um conjunto de sensores do tipo sonar, gerando de modo autônomo os comandos de aceleração e de giro de direção, de modo a estacionar o carro em uma vaga paralela. O sistema conta com um controlador implementado através do uso de um autômato finito baseado em regras, e os resultados obtidos demonstram que o controlador é capaz de estacionar corretamente o carro baseado apenas nas informações provenientes dos sensores.

Palavras-chave: Robótica autônoma; controle sensorial-motor inteligente; condução autônoma.

1 Introdução

Os veículos autônomos (RMA - Robôs Móveis Autônomos) tem atraído a atenção de um grande número de pesquisadores, devido ao desafio que este novo domínio de pesquisas propõe: dotar sistemas de uma capacidade de raciocínio inteligente e de interação com o meio em que estão inseridos. Os RMA's podem perceber o ambiente em que estão inseridos através da leitura de seus sensores (infravermelho, Sonar, lasers, câmeras de vídeo, etc), e através desta percepção sensorial eles podem planejar melhor as suas ações [1, 2].

Atualmente os robôs móveis atuam em diferentes áreas, como desarmamento de bombas, exploração de ambientes hostis, e a condução de veículos robotizados. Alguns exemplos de RMA's são: o sistema desenvolvido pelo NavLab da CMU [3, 4] que é capaz de conduzir uma caminhonete pelas estradas americanas; os robôs do tipo rover enviados para Marte pela NASA [5]; o robô Dante, que explora o interior de vulcões [6]; e o sistema de controle de um veículo Ligier elétrico desenvolvido pelos pesquisadores do INRIA na França [7, 8]. Todos esses sistemas possuem em comum a capacidade de receber leituras de sensores que lhes dão informações sobre o ambiente em que estão inseridos, e de modo semi ou completamente autônomo geram os comandos que fazem com que eles se desloquem no ambiente de modo seguro, ou seja, sem se chocar contra obstáculos ou colocar em risco a sua integridade ou a dos diferentes elementos presentes no ambiente.

A partir de estudos e trabalhos de pesquisa desenvolvidos pelo Grupo de Inteligência Artificial do PIPCA[1], foram criadas as bases para o desenvolvimento de aplicações na área de robótica autônoma móvel. Destaca-se particularmente o desenvolvimento do sistema SEVA (Simulador de Estacionamento de Veículos Autônomos) [9], que realiza a tarefa de estacionamento de um robô

[1]http://inf.unisinos.br/ osorio/gia.html

não-holonômico (tipo carro) em uma vaga paralela. Este simulador utiliza um ambiente bidimensional (2D) no qual um veículo equipado com seis sensores do tipo infra-vermelho é controlado através do uso de um autômato finito (SEVA-A) ou através de uma Rede Neural (SEVA-N) do tipo *Jordan Cascade-Correlation* (J-CC) [10, 11].

Uma das limitações do SEVA original é que pelo fato do ambiente simulado ser bidimensional, os objetos presentes não possuem altura, o que torna o modelo muito simplificado em relação à realidade. Em um ambiente tridimensional (3D), tarefas como a localização do meio-fio da calçada são mais difíceis de serem executadas, pelo fato deste possuir uma altura muito pequena (em média 15cm). Outra limitação do modelo é que neste ambiente bidimensional, a presença de ruídos nos dados coletados por sensores infravermelhos era praticamente nula, pois os sensores simulados eram muito simples, o que tornava o modelo muito diferente da realidade.

Devido a estas limitações, foi proposto o desenvolvimento do sistema SEVA3D (Simulador de Estacionamento de Veículos Autônomos em um ambiente tridimensional), que realiza a tarefa de estacionamento de um veículo em uma vaga paralela utilizando um modelo tridimensional de ambiente e sensores, portanto muito mais próximo à realidade. Este sistema utiliza o simulador SimRob3D [12], e realiza o estacionamento do veículo de forma autônoma através de um autômato finito baseado em regras. Sensores do tipo Sonar (3D) foram instalados em pontos estratégicos do veículo para que a tarefa de estacionamento fosse possível.

Nas seções seguintes serão descritos o modelo de simulação adotado, o controlador baseado em regras, os experimentos realizados, os resultados obtidos, e ao final serão apresentadas melhorias que poderão vir a ser implementadas futuramente.

2 Trabalhos relacionados

Estudos relativos ao estacionamento de veículos de forma autônoma vem sendo realizados por diversos pesquisadores, dentre os quais podemos destacar os estudos realizados no INRIA [7, 8] para o controle e estacionamento de um veículo Ligier elétrico de forma autônoma. O veículo foi equipado com 14 sensores do tipo Sonar. A Figura 1 mostra um esquema do modelo.
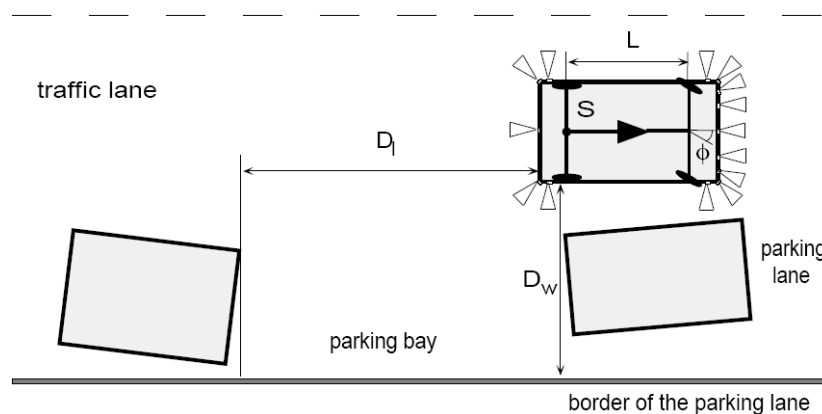
Figura 1: Modelo do INRIA [8]

O sistema em questão utilizou o modelo de cinemática Ackerman [13] para descrever o deslocamento do veículo. A tarefa de estacionamento foi dividida em três etapas: localização da vaga; ajuste da posição do veículo para permitir o estacionamento; manobra de estacionamento. Enquanto

o veículo percorre a via à procura de uma vaga, um mapa do ambiente vai sendo elaborado a partir dos dados sensoriais. Ao ser localizada uma vaga de tamanho suficiente, o sistema calcula a posição de início da manobra e movimenta o veículo até esta posição. Após o veículo estar corretamente posicionado, começa a manobra de estacionamento propriamente dita, que é realizada através do uso de funções senoidais (descritas em [14]), que fazem com que o veículo percorra uma trajetória suave, como mostra a Figura 2.
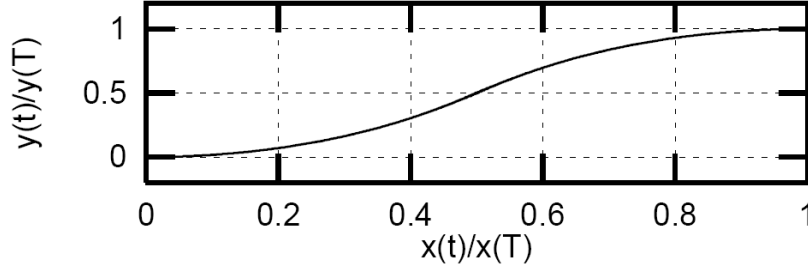
Figura 2: Trajetória percorrida [8]

Durante toda manobra de estacionamento, o veículo é controlado de forma autônoma usando um algoritmo iterativo, e a cada instante o ajuste da velocidade e do giro da direção são calculados pelas fórmulas:

$$\phi(t) = \phi_{max}\, k_\phi\, A(t), \quad 0 \le t \le T, \tag{1}$$

$$v(t) = v_{max}\, k_v\, B(t), \quad 0 \le t \le T, \tag{2}$$

onde $\phi(t)$ é o giro da direção no instante t, $v(t)$ é a velocidade do veículo no instante t, T é a duração máxima da manobra, ϕ_{max} é o giro máximo da direção, e v_{max} é a velocidade máxima durante a manobra. $k_\phi = \pm 1$ indica se o estacionamento é em uma vaga à esquerda (-1) ou à direita $(+1)$, e $k_v = \pm 1$ indica se a direção do movimento é para a frente $(+1)$ ou para trás (-1). Os valores de $A(t)$ e $B(t)$ são calculados através das fórmulas:

$$A(t) = \begin{cases} 1, & 0 \le t < t', \\ \cos \frac{\pi(t-t')}{T^*}, & t' \le t \le T - t', \\ -1, & T - t' < t \le T, \end{cases} \tag{3}$$

$$B(t) = 0.5(1 - \cos \tfrac{4\pi t}{T}), \quad 0 \le t \le T, \tag{4}$$

$$t' = \tfrac{T-T^*}{2}, \quad T^* < T, \tag{5}$$

onde T (duração da manobra) e T^* (duração da parte curva da manobra) são estimados a partir das distâncias da largura da vaga (Di) e da profundidade da vaga (Dw).

Para que o cálculo da profundidade da vaga fosse possível, foi necessária a instalação de uma barreira de altura moderada junto ao meio-fio, para que os sensores pudessem detectá-lo com mais precisão [7]. A vantagem de utilizar funções senoidais ao invés de um autômato finito baseado em regras é que os movimentos tornam-se muito mais suaves, ou seja, não há uma transição brusca de estados. As principais desvantagens desta abordagem são a necessidade da instalação de uma barreira, o que torna o estacionamento inviável em vias convencionais, e a necessidade da utilização de muito mais sensores do tipo sonar (quatorze no total). No SEVA3D, foram necessários apenas seis sensores.

3 Simulador SEVA3D

O simulador SEVA3D possui diversas melhorias em relação ao SEVA original (2D), dentre as quais é possível destacar:

- Utiliza um ambiente tridimensional e sensores mais realistas, que simulam inclusive a presença de ruído;
- Realiza o estacionamento em uma vaga paralela independente da presença de outros carros estacionados (o sistema funciona mesmo não haja nenhum outro carro);
- É muito mais robusto em relação à distância entre o carro e o meio-fio, aceitando valores de 2 e 4 metros;
- Realiza o afastamento do veículo de forma automática se ele estiver muito próximo aos carros estacionados antes do início da manobra (distâncias menores que 30cm em relação aos carros à direita);
- Permite que a manobra seja visualizada sob qualquer ângulo e posição do ambiente tridimensional.

Os principais componentes do modelo do simulador SEVA3D são:

- Modelo sensorial;
- Modelo cinemático (deslocamento do veículo);
- Comandos relacionados ao deslocamento (avançar, recuar e velocidade) e ao giro do veículo (rotação da direção);
- Sistema de controle do veículo durante o estacionamento (autômato finito).

3.1 SimRob3D

Para a implementação do SEVA3D foi utilizado o simulador SimRob3D [12]. Este simulador tem como principal característica o fato de utilizar um ambiente tridimensional para a navegação dos robôs móveis simulados. O ambiente pode ser modelado em diversos softwares de modelagem tridimensional existentes no mercado, como o 3D Studio MAX[2], e permite que sejam especificados os diversos elementos presentes em um ambiente (objetos, luzes, texturas), o que resulta em um nível de realismo muito superior aos obtidos em simuladores bidimensionais. O simulador possui diferentes modelos sensoriais e cinemáticos, permitindo a configuração de diversos tipos de robôs.

O SimRob3D não possui internamente nenhuma forma de controle da simulação, pois todo o controle da simulação, inserção de robôs e sensores, é realizado por um controlador. Este controlador é externo ao SimRob3D, e a interface entre os dois softwares é realizada utilizando-se uma biblioteca dinâmica (DLL). O SimRob3D disponibiliza uma API *(Application Programming Interface)* que fornece diversos recursos para que o controlador, no caso o SEVA3D, possa interagir com ambiente da simulação. Outro ponto a ser destacado é que o controlador não possui acesso a nenhuma informação relativa ao ambiente, a não ser as fornecidas pelos sensores instalados no veículo.

3.2 Modelo sensorial

Os sensores de distância simulam sensores do tipo Sonar [13], sendo capazes de estimar a distância entre o veículo e os obstáculos presentes no ambiente: outros carros e a calçada. Os seis sensores utilizados estão distribuídos em pontos estratégicos do carro, como mostra a Figura 3. Foram implementados apenas os sensores da lateral direita do veículo, pois o modelo se restringiu

[2] http://www.autodesk.com/3dsmax/
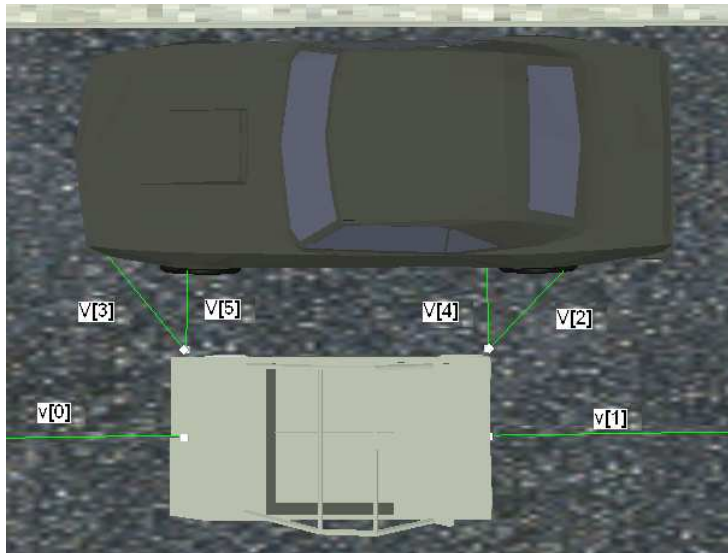
Figura 3: Localização dos sensores

apenas ao estacionamento em vagas paralelas localizadas no lado direito do carro, caso típico em pistas de duas vias. Os sensores **V[2]** e **V[3]** foram instalados com uma certa inclinação em relação ao solo, para que seja possível detectar o meio-fio.

Os sonares são sensores muito úteis em robótica, pois podem fornecer estimativas robustas relativas à distância de vários tipos de objetos. Estas estimativas podem ser utilizadas para perceber o ambiente, e geralmente são difíceis de serem obtidas através de outros métodos. O modelo utilizado pelo SimRob3D para simular os sensores do tipo sonar é estocástico. Diversos raios são traçados da posição do sensor em direção a sua orientação. Uma técnica de RayCast[3] foi utilizada para traçar os raios, que são distribuídos aleatoriamente na forma de um cone, e se algum deles colidir com alguma das faces tridimensionais do ambiente, a distância até o ponto de colisão é informada.

Devido ao fato das distâncias informadas pelos sensores serem estocásticas, foi utilizado um método de *janelamento* [16], com uma janela de tamanho 10, para a leitura dos dados sensoriais. Além dos sensores do tipo Sonar, é utilizado um odômetro para verificar se uma vaga é suficientemente grande para permitir o estacionamento do veículo.

3.3 Modelo cinemático

A movimentação do veículo respeita o modelo Ackerman de cinemática de um veículo [13], também adotado em [17]. Neste modelo é simulado um veículo representado por um volume retangular suportado por quatro rodas, onde as rodas traseiras possuem um eixo fixo e as rodas dianteiras podem ser direcionadas, através do giro da barra da direção. As coordenadas do veículo são definidas por $P(t) = (x(t), y(t), \theta(t))$, onde $x(t)$ e $y(t)$ definem o ponto médio do eixo traseiro do veículo (ponto 0) no instante t e $\theta(t)$ indica a sua orientação (ângulo em relação à direção de referência). O

[3]RayCast é uma técnica da computação gráfica que simula os efeitos físicos associados com a propagação de raios de luz [15]

deslocamento do veículo é estimado através das seguintes equações [13]:

$$x(t) = \int_0^t V(t)\cos[\phi(t)]\cos[\theta(t)]\,dt \tag{6}$$

$$y(t) = \int_0^t V(t)\cos[\phi(t)]\sin[\theta(t)]\,dt \tag{7}$$

$$\theta(t) = \int_0^t \frac{V(t)}{L}\,\tan[\phi(t)]\,dt \tag{8}$$

onde $V(t)$ representa a velocidade do veículo no instante t, $\phi(t)$ representa o giro da direção do veículo no instante t, e L indica o comprimento do eixo das rodas. A Figura 4 apresenta um esquema
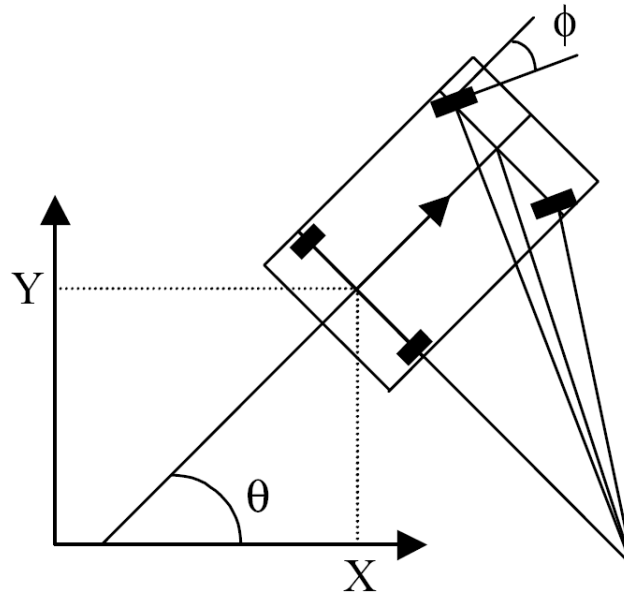
Figura 4: Modelo cinemático

do modelo cinemático adotado no SEVA3D. Convém notar que os valores $x(t)$ e $y(t)$ dependem de $\theta(t)$, que é alterado de forma contínua durante o deslocamento do veículo.

3.4 Controle

O deslocamento do veículo é obtido na simulação através do controle de sua velocidade V e do giro da direção ϕ. Diferentemente do modelo do SEVA original, no SEVA3D a velocidade adotada durante a manobra pode ser definida pelo usuário, em valores de 0 (parado) a 80 (muito rápido). Quando o veículo precisa se deslocar para trás, é a velocidade utilizada com sinal negativo.

O giro da direção durante a manobra (ϕ) também pode ser informado pelo usuário, em valores entre 0 e 35, que correspondem ao giro máximo da direção em graus durante a entrada do carro na vaga. Quando o veículo precisa virar à direita, ϕ é utilizado com sinal negativo.
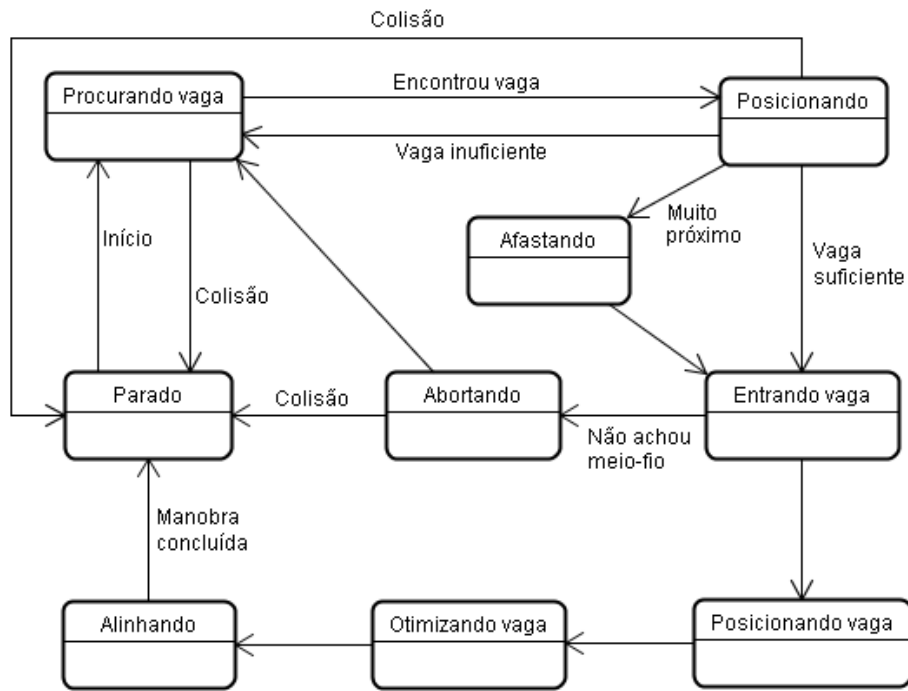
Figura 5: Estados do autômato

3.5 Autômato finito

A Figura 5 mostra o diagrama de estados do autômato finito utilizado pelo SEVA3D. Foram utilizados nove estados, dois a mais que no SEVA original. As finalidades de cada um dos estados são:

- **Parado:** Estado inicial do autômato, antes do veículo começar a se deslocar. Também é o estado atingido quando se detecta uma colisão iminente;
- **Procurando vaga:** Início da manobra de estacionamento, quando o veículo se desloca para frente com a direção reta, buscando encontrar uma vaga. Se encontrar, o estado muda para **Posicionando**;
- **Posicionando:** O veículo se desloca para a frente, a fim de atingir uma posição que torne possível o estacionamento, e também para verificar se a vaga é suficientemente grande. Se não for, o estado volta para **Procurando vaga**. Caso o veículo se encontre muito próximo aos carros estacionados à direita, o estado muda para **Afastando**. Se a vaga for de tamanho suficiente, o estado muda para **Entrando vaga**;
- **Afastando:** O veículo se encontra muito próximo aos carros estacionados na lateral direita (menos de 30cm), o que dificulta a manobra. Neste caso, o carro inicialmente se desloca para a frente com a direção voltada para esquerda, a fim de se afastar dos carros estacionados. Após ter se afastado o suficiente, o carro é novamente alinhado de forma paralela em relação à via.

Para que isto ocorra, o giro da barra de direção é calculado pela fórmula:

$$\phi(t) = \arctan\left(\frac{L\,\theta(t)}{V(t)}\right) \tag{9}$$

Se $\phi(t)$ for maior que o giro máximo da barra de direção (ϕ_{max}), o ajuste é limitado a(ϕ_{max}), o que faz com que a manobra exija vários passos sucessivos até ser concluída. Após o carro estar alinhado, ele volta para trás com a direção reta até atingir a posição ideal para o início da manobra de estacionamento, e o estado muda para **Entrando vaga**;

- **Entrando vaga:** O carro vira a direção para a direita e começa a se movimentar para trás, de forma a entrar na vaga. Os valores de ϕ e V utilizados são os que foram definidos empiricamente pelo usuário. Quando o sensor **V[2]** (Figura 3) detectar o meio-fio da calçada, o estado muda para **Posicionando vaga**. Caso o veículo ultrapasse uma distância limite sem encontrar o meio-fio, o estado muda para **Abortando**;

- **Abortando:** Quando não for possível encontrar o meio-fio da calçada, o veículo faz uma manobra de retorno até o meio da via, onde o giro da direção é calculado pela fórmula:

$$\phi(y) = \int_{y_0}^{y_d} -\theta(y)\left(\frac{y - y_d}{y_0 - y_d}\right) dy \tag{10}$$

  onde y_0 é a posição carro em relação a y no início deste estado, y_d é a posição y desejada (no meio da via). Esta fórmula faz com que a direção do carro inicialmente se volte totalmente para a direita (ϕ_{max} à direita) e a medida que o carro vai se aproximando de y_d, as rodas do veículo tendem a se posicionar de forma paralela à via, fazendo com que a manobra de retorno seja suave. Quando $y \geq y_d$, o estado volta para **Procurando vaga**;

- **Posicionando vaga:** neste estado, o veículo prossegue para trás, mas com a direção à esquerda. Quando o sensor **V[3]** detectar o meio-fio da calçada ou o sensor **V[1]** detectar um obstáculo a menos de 30cm, o estado muda para **Otimizando vaga**;

- **Otimizando vaga:** o carro se desloca para frente de forma a ficar paralelo ao meio-fio, e o giro da direção é calculado novamente usando a Fórmula (9). Após o carro estar paralelo, o estado muda para **Alinhando**;

- **Alinhando:** neste estado o veículo é movimentado de forma a ficar a uma distância razoável em relação aos carros estacionados à frente ou atrás. Após o alinhamento estar concluído, o estado muda para **Parado** e a manobra é encerrada com sucesso.

4 Implementação

Para a implementação do controlador autônomo, foi utilizado o simulador SimRob3D [12], e foi construído no *software* 3D Studio Max um modelo de tridimensional de uma via com carros estacionados, para que fosse possível realizar o estacionamento de forma adequada. A Figura 6 mostra uma imagem do ambiente modelado. O veículo modelado para realizar o estacionamento é uma reprodução de um veículo do tipo Buggy, pertencente ao curso de engenharia elétrica da UNISINOS[4]. O simulador implementado faz uma integração discreta do modelo descrito nas fórmulas 6, 7, 8, 9 e 10.

[4]http://www.exatec.unisinos.br/ autonom/
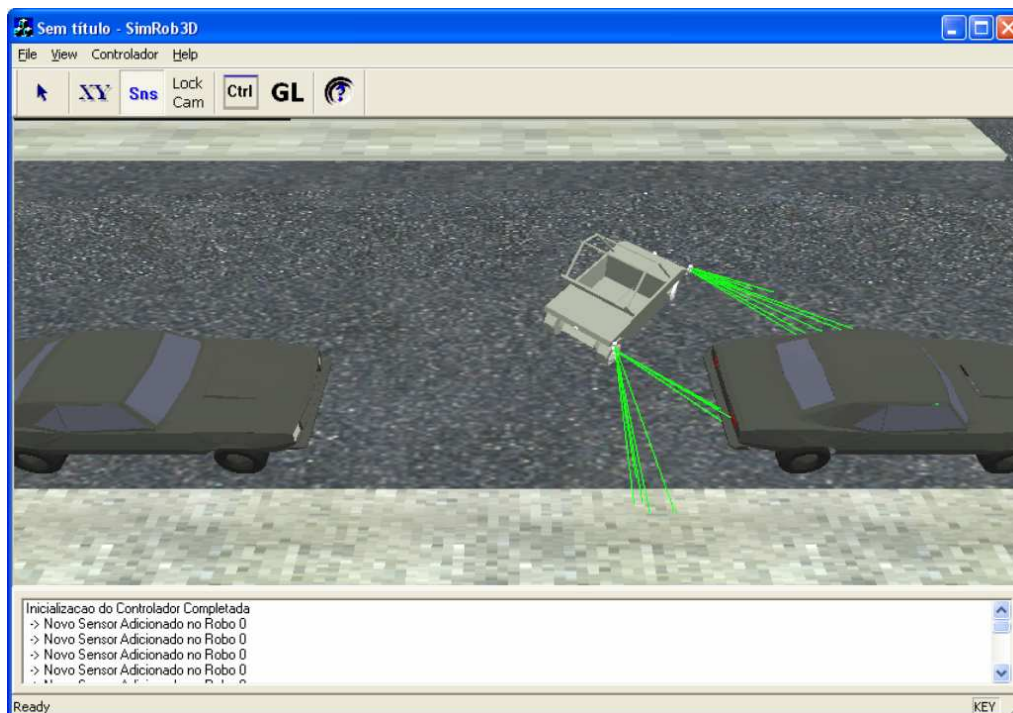
Figura 6: Ambiente modelado

Para a implementação do autômato finito, foi desenvolvido um controlador do SimRob3D *(control.dll)* em linguagem C, que implementa toda a funcionalidade do SEVA3D[5] relativa ao controle e a movimentação dos veículos. Para o acompanhamento da manobra de estacionamento, o usuário pode visualizar o ambiente com a câmera virtual posicionada em qualquer ponto, e além disto, uma tela fornece diversas informações relativas à simulação, como o estado do autômato, os valores percebidos pelos sensores, a velocidade, giro da direção, etc. A Figura 7 mostra a tela contendo estas informações. Vários testes preliminares foram realizados, e se verificou que o modelo se comportava de forma bastante similar à realidade.

5 Resultados

Para a validação do SEVA3D, diversos experimentos foram planejados e realizados. Para se verificar a robustez do sistema, foram realizadas diferentes simulações, variando os seguintes fatores em cada uma delas:

- A posição inicial do carro em relação ao centro da via. Foram testadas posições desde poucos centímetros em relação aos demais carros estacionados até mais de 3 metros destes;
- O estacionamento foi testado entre dois carros, na presença de apenas um carro estacionado e também sem nenhum outro carro estacionado na via;
- O estacionamento foi testado próximo à esquinas e entradas de garagem, onde o referencial do meio fio não existia à frente ou atrás da vaga.

[5] Tanto o SimRob3D quanto o SEVA3D estão disponíveis para download no site http://inf.unisinos.br/ osorio/seva3d/

Figura 7: Tela de informações

Devido a natureza estocástica dos dados sensoriais, foram planejados 10 experimentos utilizando sementes aleatórias diferentes para cada uma das configurações de simulação, e ao final foi calculada a média e o desvio padrão dos resultados obtidos.

Em todas estas situações, o SEVA3D foi capaz de estacionar o veículo de forma plenamente satisfatória, com uma média de $26,16cm$ em relação ao meio fio e um desvio padrão de $5,92cm$, o que demonstra que o sistema é seguro e robusto no estacionamento de veículos.

6 Conclusões e perspectivas

Este trabalho teve como objetivo o desenvolvimento de um simulador para o controle de veículos autônomos em uma tarefa de estacionamento em vagas paralelas, utilizando um ambiente tridimensional e sensores do tipo Sonar. Os resultados obtidos nas simulações realizadas com a ferramenta desenvolvida, o SEVA3D, demonstraram que o sistema de controle possui a capacidade de controlar corretamente o veículo, cumprindo seu objetivo principal: estacionar o veículo de modo autônomo corretamente na vaga, sem bater nos demais obstáculos que estão ao seu redor. As verificações, tanto numéricas quanto visuais, permitiram constatar que a tarefa pôde ser corretamente executada na maioria das simulações realizadas, demonstrando que o estacionamento é estável, seguro e robusto.

No futuro melhorias como a utilização de uma Rede Neural [16] e Lógica *Fuzzy* para a implementação do autômato podem vir a ser adotadas, assim como a implementação do SEVA3D em um veículo real.

Referências

[1] MEDEIROS, A. Introdução a robótica. *ENA-98 Encontro Nacional de Automática - 50º Congresso da SBPC*, p. 56–65, 1998.

[2] HEINEN, F. J. *Robótica Autônoma: Integração entre Planificação e Comportamento Reativo*. São Leopoldo, RS: UNISINOS Editora, 1999.

[3] POMERLEAU, D. Neural network based autonomous navigation. *Vision and Navigation - The CMU Navlab. Kluwer Academic Publihers*, 1990.

[4] BATAVIA, P.; POMERLEAU, D.; THORPE, C. *Applying Advanced Learning Algorithms to ALVINN*. Pittsburgh, 1996.

[5] STONE, H. W. Mars pathfinder microver: A low-cost, low-power spacecraft. *Proceeding of the 1996 AIAA - Forum on advanced developments in Space Robotics*, August 1996.

[6] LEMONICK, M. Dante tours the inferno. *Time Magazine - Time Domestic/Science*, v. 144, n. 7, 1994.

[7] PAROMTCHIK, I. E.; LAUGIER, C. Autonomous parallel parking of a nonholonomic vehicle. *Procedings of the IEEE International Symposium on Intelligent Vehicles*, p. 13–18, September 1996.

[8] LAUGIER, C. et al. Sensor based control architeture for a car-like vehicle. *Proceedings of the 1998 IEEE/RSJ Int. Conf. of Intelligent Robots and Systems*, October 1998.

[9] OSORIO, F. S.; HEINEN, F. J.; FORTES, L. Controle inteligente de veículos autônomos: Automatização do processo de estacionamento de carros. *Anais do SEMINCO 2001 - FURB*, Outubro 2001.

[10] FAHLMAN, S. E.; LEBIERE, C. *The Cascade-Correlation Learning Architecture*. [S.l.], 1990.

[11] OSORIO, F. S.; HEINEN, F. J.; FORTES, L. Controle da tarefa de estacionamento de um veículo autônomo através do aprendizado de um autônomo finito usando uma rede neural J-CC. *Anais do SBRN 2002 - Porto de Galinhas, PE*, 2002.

[12] HEINEN, F. J. *Sistema de Controle Híbrido para Robôs Móveis Autônomos*. Dissertação (Dissertação de Mestrado) — Computação Aplicada, Unisinos, São Leopoldo, RS, 2002.

[13] DUDEK, G.; ZHANG, C. *Computational Principles of Mobile Robotics*. Cambridge, UK: Cambridge University Press, 2000.

[14] MURRAY, R.; SASTRY, S. Steering nonholonomic systems using sinusoids. *Proc. of IEEE Int. Conf. On Decision and Control*, p. 2097–2101, December 1990.

[15] FOLEY, J. D. *Introduction to Computer Graphics*. [S.l.]: Addison-Wesley, xxviii, 1994.

[16] HAYKIN, S. *Redes Neurais: Princípios e Prática*. 2. ed. [S.l.]: Bookman, 2001.

[17] GARNIER, P. et al. Motion autonomy through sensor-guided manoeuvres. *Procedings of the Intelligent Cars and Automated Highway Systems - IEEE-RSJ International Conference on Intelligent Robots and Systems Workshop*, September 1999.

Implementação de Protocolos de Interação no Ambiente SACI

Issao Hirata (LTI/POLI/USP)
issao.hirata@poli.usp.br

Jomi Fred Hübner (DSC/FURB)
jomi@inf.furb.br

Jaime Simão Sichman (LTI/POLI/USP)
jaime.sichman@poli.usp.br

Resumo. O ambiente SACI (Simple Agent Communication Infrastructure) (SACI, 2004) é uma ferramenta para a construção de Sistemas Multiagentes. O ambiente proporciona uma infra-estrutura que facilita a implementação de agentes, permitindo que estes se conheçam e se comuniquem de forma transparente. Este trabalho teve como objetivo aperfeiçoar este ambiente através da implementação de uma nova funcionalidade. Trata-se do desenvolvimento uma biblioteca que permite ao ambiente reconhecer e monitorar os protocolos comunicação que regem os mecanismos de interação entre os agentes.

Palavras-chave: Comunicação entre Agentes; protocolos de comunicação entre Agentes; SACI.

1 Introdução

Um Sistema Multiagentes é uma técnica de Inteligência Artificial Distribuída em que é utilizado um conjunto de elementos, chamados agentes, que têm características como autonomia, pró-atividade, reatividade, deliberação, etc**.** (Wooldridge, M, 2002).

A técnica de Sistemas Multiagentes tem sido utilizada nos mais variados domínios e tem se mostrada adequada para a resolução de certos problemas computacionais em que as técnicas tradicionais de computação não são aplicáveis. Desta forma, existe um grande interesse, tanto acadêmico como comercial, no aperfeiçoamento desta tecnologia.

Diante disto, surgiu a necessidade de ferramentas que facilitem o desenvolvimento de tais sistemas. O ambiente SACI (Simple Agent Communication Infrastructure) (SACI, 2004) é uma destas ferramentas e fornece uma infra-estrutura para a construção de Sistemas Multiagentes.

Como em um Sistema Multiagentes não existe um controle central nem um armazenamento global de informações, os agentes não têm capacidade ou informação suficiente para resolverem os problemas sozinhos. Neste contexto, faz-se necessário à existência de algum tipo de interação entre estes agentes para que o grupo como um todo consiga resolver o problema.

Percebe-se então a necessidade de se definir protocolos que estabeleçam uma comunicação confiável entre estes agentes. Tais protocolos, também chamados *protocolos de interação* entre agentes, definem regras e estruturas que estabelecem um mecanismo para que a comunicação entre estes agentes seja objetiva e eficiente.

O objetivo deste trabalho consistiu na inserção de mecanismos que possibilitam ao ambiente SACI reconhecer e implementar tais protocolos. A arquitetura interna do ambiente foi modificada acrescentando novos componentes que permitem ao SACI interpretar estes protocolos e gerenciar as interações entre os agentes.

Nas próximas seções, são apresentados maiores detalhes sobre o trabalho. Na seção 2, são discutidos alguns detalhes referentes à padronização de linguagens de comunicação entre agentes.

A seção 3 apresenta de forma resumida o ambiente SACI. As duas seções a seguir se referem a detalhes de implementação do trabalho. Enquanto a seção 4 descreve a linguagem que foi definida para estruturar tais protocolos, a seção 5 detalha a implementação dos componentes na arquitetura do SACI. A seção 6 ilustra o funcionamento do ambiente, através da utilização de um exemplo utilizando o protocolo *Contract Net* (Smith, 1980), que serve para realizar uma alocação dinâmica de tarefas entre agentes. Finalmente, na seção 7 apresentam-se as conclusões deste trabalho e algumas extensões que podem lhe ser incorporadas no futuro.

2 Comunicação entre agentes

Em um Sistema Multiagentes nenhum agente possui conhecimento, capacidade ou condição de resolver um problema sozinho, a comunicação entre os agentes é um dos aspectos mais importantes no desenvolvimento de um SMA. Neste sentido é necessário que estes agentes disponham de métodos eficientes para que possam se conhecer e trocar informações.

Para tentar garantir com que as trocas de mensagens entre estes agentes ocorram de maneira confiável e eficiente, surgiram então os conceitos de Linguagem de Comunicação entre Agentes (LCA) e os Protocolos de Interação.

2.1 Linguagens de comunicação entre agentes

O estabelecimento de uma Linguagem de Comunicação entre Agentes (LCA) é fundamental para garantir que as mensagens trocadas possam ser entendidas e interpretadas de maneira comum a todos os agentes envolvidos. Uma LCA fornece uma estrutura que permite aos agentes a troca de conhecimento e não somente de dados.

Para tanto, ela estabelece uma sintaxe para a composição de mensagens. Esta sintaxe define os campos que a mensagem deve possuir e a estrutura que ela deve obedecer. Além da sintaxe, uma LCA também define uma semântica (o que as informações representam) e uma pragmática (como as informações devem ser interpretadas), associando assim um significado para cada mensagem.

Existem várias propostas de LCAs, porém duas delas possuem maior destaque: o KQML (Labrou e Finin, 1997) e a FIPA ACL (FIPA, 2002b).

KQML é a uma especificação é uma LCA baseada nos princípios da teoria dos atos de fala (Searle, 1970). A estrutura básica de uma mensagem KQML foi definida de uma maneira simples que pode ser facilmente analisada tanto por uma pessoa quanto por um parser. A sintaxe básica de uma estrutura KQML pode ser vista na Figura 1.

A FIPA (Foundation for Intelligent Physical Agentes) também especifica uma Linguagem de Comunicação entre Agentes denominada FIPA ACL (FIPA, 2002b). Esta linguagem, assim como o KQML também está baseado no princípio de atos de fala.

Uma diferença entre KQML e FIPA ACL pode se encontrar na estrutura da sintaxe da mensagem. Ao contrário do KQML, a estrutura da mensagem FIPA ACL prevê alguns campos que podem ser utilizados para a realização do controle de conversas, como por exemplo **conversation-id e protocol**. Estes campos são utilizados pelos Protocolos de Interação.

```
(KQML-performative
        :sender     <word>
        :receiver   <word>
        :language   <word>
        :ontology   <word>
        :content    <expression>
        ...)
```

Parameter	Category of Parameters
performative	Type of communicative acts
sender	Participant in communication
receiver	Participant in communication
reply-to	Participant in communication
content	Content of message
language	Description of Content
encoding	Description of Content
ontology	Description of Content
protocol	Control of conversation
conversation-id	Control of conversation
reply-with	Control of conversation
in-reply-to	Control of conversation
reply-by	Control of conversation

Figura 1: Estruturas das sintaxes de mensagens KQML (esquerda) e FIPA ACL (direita)

2.2 Protocolos de interação entre agentes

Enquanto as LCAs são responsáveis por garantir meios para que os agentes possam trocar uma mensagem, os protocolos de interação entre agentes (ou protocolos de comunicação) são responsáveis por governar e gerenciar as trocas de mensagem dentro de uma conversa.

Uma conversa é formada por um conjunto de mensagens trocadas entre os agentes com o propósito de se alcançar um determinado objetivo. A utilização de um protocolo de interação aumenta o desempenho desta comunicação já que este permite o estabelecimento de objetivos em comum e determinação das tarefas conjuntas, evitando-se assim conflitos desnecessários.

Os protocolos de interação normalmente obedecem a certos padrões de interação. Alguns destes padrões são: cooperação (os agentes se agrupam e trabalham visando um objetivo comum), coordenação (os agentes se agrupam de modo a explorar interações benéficas e a evitar as prejudiciais) ou negociação (os agentes interagem procurando chegar a um acordo que seja aceitável para todas as partes envolvidas). Estes padrões de interação também são conhecidos como domínio do protocolo de interação.

Na literatura são encontradas algumas propostas para a especificação dos protocolos de informação, como a que é encontrada em (Populaire et al., 1993). Porém, uma proposta mais recente, recomendada pela FIPA (Foundation for Intelligent Physical Agents) tem se tornado um padrão para a representação destes protocolos de interação. Trata-se da AUML (Odell et al., 2001), uma adaptação da técnica de modelagem de orientação a objetos UML (Unified Modeling Language). Um dos diagramas da AUML, que descreve um protocolo chamado Contract Net, pode ser visto na figura 2.

Infelizmente, a AUML estabelece apenas técnicas gráficas (diagramas) para representar os protocolos de interação, não existindo ainda nenhuma sintaxe que torne as informações manipuláveis computacionalmente.
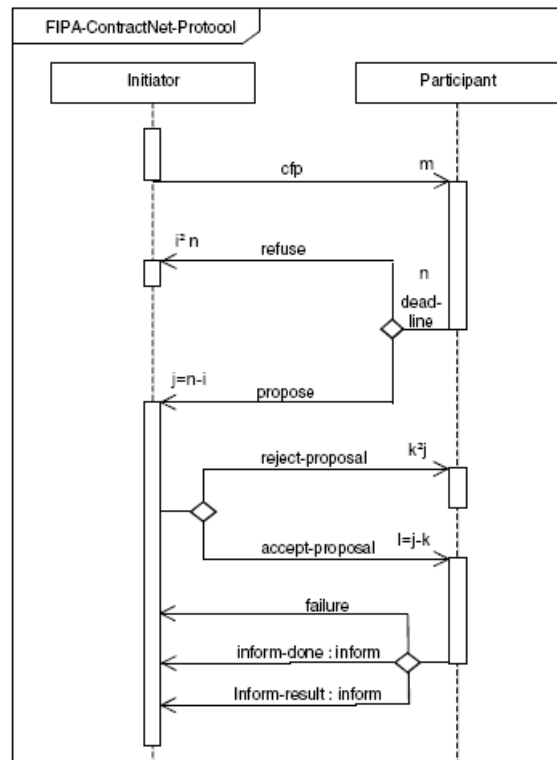
Figura 2: Exemplo de Diagrama de Seqüências AUML - Contract Net

3 Ambiente SACI

O SACI (Simple Agent Communication Infrastucture) (SACI, 2004) é uma ferramenta desenvolvida no Laboratório de Técnicas Inteligentes da USP (LTI/EPUSP) e se trata de um ambiente para a construção e execução de Sistemas Multiagentes.

O ambiente provê uma infra-estrutura que serve de base para a implementação do comportamento social dos agentes, fornecendo meios para que estes possam se conhecer e se comunicar de maneira transparente. Basicamente, o ambiente fornece dois recursos principais: uma API (Application Program Interface) para compor, enviar e receber mensagens, e ferramentas que auxiliam o projetista a superar dificuldades em relação à programação em ambientes distribuídos (Hübner and Sichman, 2000). As principais características do ambiente são:

- os agentes são agrupados em sociedades e utilizam KQML para se comunicar. O ambiente possui funções para compor, enviar e receber mensagens KQML;
- os agentes são identificados pelo nome, sua localização na rede é transparente e as mensagens são transportadas utilizando-se o nome do receptor;
- conforme recomendado pela arquitetura KQML, existe um facilitador que possui um serviço de páginas amarelas;
- o ambiente foi desenvolvido através da linguagem JAVA, garantindo assim a sua portabilidade.

Para o envio e recebimento de mensagens KQML entre os agentes, foi desenvolvida uma API (Application Program Interface) (Hübner e Sichman, 2000). Esta API é implementada no SACI a

partir de um componente chamado **MBox** (ou MessageBox), que funciona como interface de comunicação entre o agente e o ambiente.

O **MBox** fornece métodos para a composição, envio (síncrono e assíncrono) e recebimento de mensagens KQML. Além disso, o **MBox** também permite que o agente anuncie suas habilidades e consulte os serviços prestados por outros agentes.

Cada agente do sistema possui o seu próprio **MBox** associado. Este **MBox** servirá como porta de Entrada e Saída para a troca de informações entre este agente e o restante da sociedade. Existe ainda um agente especial, denominado **Facilitator**, que tem a função de regular a comunicação da sociedade, registrar as funcionalidades (páginas amarelas) e agentes (páginas brancas) ativos.

Como dito anteriormente, o ambiente SACI fornece a funcionalidade de troca de mensagens transparente para os agentes de uma sociedade, porém de forma não estruturada. Para isto, torna-se necessário especificar uma linguagem de descrição de protocolos (LDP), bem como uma arquitetura subjacente que possa processar tal linguagem. Tais aspectos, objetivos principais deste trabalho, estão descritos nas seções 4 a 6.

4 Especificação da linguagem de descrição dos protocolos

A especificação da linguagem de descrição de Protocolos que será utilizada foi elaborada baseando-se nos diagramas da AUML e na proposta descrita em (Populaire et al., 1993), possuindo a estrutura mostrada na Figura 3.
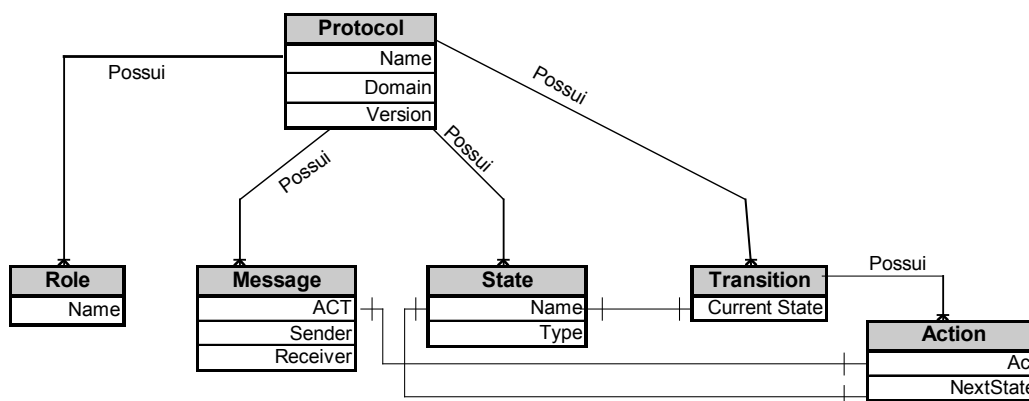
Figura 3: Diagrama de Relacionamentos da Estrutura de Dados utilizada para a Especificação dos Protocolos

Os principais elementos desta estrutura são os seguintes:

1. Protocol: é o elemento raiz dessa estrutura de dados e é formado por elementos do tipo ***Role***, ***Message***, ***State*** e ***Transitions***, definidos nesta ordem. Além disso, o ***Protocol*** também possui os atributos ***name*** - o nome do protocolo que está sendo descrito, ***domain*** – especifica o domínio do protocolo de interação e ***version*** – versão do protocolo que esta sendo usada.

2. Role: este elemento é responsável por identificar os papéis que existem dentro do protocolo. Este elemento é basicamente formado por um único atributo chamado ***name.*** Os protocolos da FIPA definem dois papéis, o ***Initiator*** e o ***Participant.***

3. Message: o elemento **Message** tem como objetivo representar as mensagens que serão trocadas durante a execução do protocolo de interação. Este elemento é formado pelos atributos ***ACT***, ***Sender e Receiver.*** O atributo ***ACT*** especifica o ato ou performativa da mensagem. Os

atributos ***Sender*** e ***Receiver*** são responsáveis por especificar o agente que envia e o agente que deve receber esta mensagem.

4. State: o elemento ***State*** armazena informações sobre os estados que existem no protocolo. O elemento ***State*** possui dois atributos necessários para definir um estado do protocolo. O primeiro atributo (name) é o nome que se deseja dar ao estado. O atributo ***type*** é responsável por representar qual o tipo do estado: **initial**, **intermediary** ou **final**.

5. Transition: o elemento ***Transition*** armazena as informações sobre as transições de estados que podem ocorrer no decorrer das interações do protocolo. Uma ***Transition*** possui um atributo chamado ***State*****,** que especifica o estado a que esta transição pertence, e um conjunto de elementos do tipo ***Action.*** Estes elementos são responsáveis por definir as possíveis ações que um agente pode tomar quando este se encontrar neste estado.

Uma ***Action*** é formada pela dupla ***ACT*** e ***NextState***, indicando quais mensagens podem ser enviadas (recebidas) e qual é o próximo estado da conversa quando tal evento ocorrer.

Para a descrição de um protocolo de interação, é utilizado um arquivo XML que obedece a estrutura definida acima. As informações descritas neste arquivo são interpretadas pelo SACI possibilitando a realização de conversas através deste protocolo. Para simplificar o processo de descrição do protocolo, foi desenvolvida uma aplicação que permite com que o projetista possa modelar o protocolo em um ambiente gráfico.

4.1 Modelagem de protocolos através da interface gráfica

Para se descrever o protocolo de comunicação na linguagem especificada acima, é necessário o conhecimento de uma série de detalhes sobre a linguagem XML e sobre como a especificação foi estruturada.

Diante desta dificuldade, surgiu a necessidade de se desenvolver um software que oferece ao programador uma interface gráfica que permita a modelagem de protocolos de uma maneira mais simples. Este software tem como objetivo gerar como saída um arquivo XML contendo a especificação do protocolo modelado de modo a auxiliar o programador, permitindo com que este não necessite de conhecer detalhes sobre a linguagem de descrição de protocolos utilizada.

Este software foi programado utilizando-se as bibliotecas gráficas da linguagem JAVA. Na figura abaixo se pode visualizar uma das telas do programa.
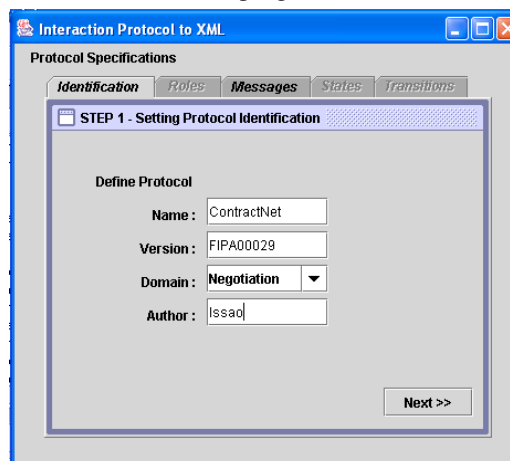
Figura 4: Exemplo de Interface do Software Gráfico

5 Implementação dos componentes no ambiente SACI

As alterações implementadas no ambiente consistem basicamente na criação de um novo componente, denominado **ProtocolBox**, que será responsável por manipular as interações de agentes de maneira análoga ao que realiza o **MBox** (já descrito anteriormente) em relação à troca de mensagens.

Este componente possui basicamente três finalidades: a de ler e interpretar as informações de cada protocolo contidas nos arquivos XML; a de gerenciar e controlar a lógica das conversas (baseadas nos protocolos); a de oferecer uma API com métodos que auxiliam a programar tais interações nos agentes.

As principais funções e métodos desta API são:

1. Iniciar uma conversa;
2. Verificar o estado do protocolo para cada agente participante;
3. Verificar as possíveis ações que o agente pode realizar;
4. Enviar e receber mensagens entre os agentes;
5. Realizar as transições de estado dos agentes de acordo com o andamento do protocolo;
6. Controle de Timeout.

O ProtocolBox é um componente que foi implementado como uma extensão do MBox. Ele contém informações de todas as conversas em que o agente está participando e oferece métodos para que ele possa manipular esta conversa. Um diagrama simplificado de como estes componentes foram implementados pode ser vista na figura 5.
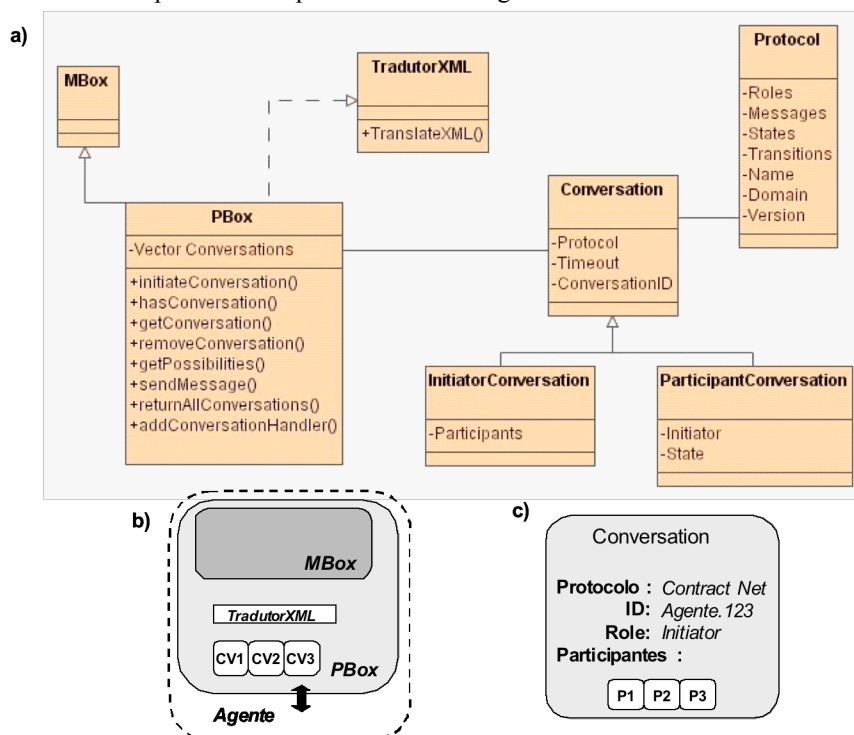
Figura 5: a) Diagrama de Classes Simplificado dos Componentes inseridos no SACI. b) Desenho esquemático do ProtocolBox c) Desenho Esquemático de um objeto Conversation

As informações de cada conversa são descritas em um objeto do tipo Conversation. que armazena dados como: estado atual da conversa, agentes envolvidos, o id da conversa e o protocolo utilizado.

6 Exemplo de funcionamento do ProtocolBox

Algumas funcionalidades do ProtocolBox podem ser mais bem entendidas através de um exemplo. Para tanto, será utilizado como protocolo de interação a Contract Net (Smith, 1980), que serve para realizar uma alocação dinâmica de tarefas entre agentes. Será utilizado para tal a versão deste protocolo proposta pela FIPA (FIPA, 2002a). A descrição AUML deste protocolo pode ser vista na figura 3 e a sua descrição em XML está representada na figura 6.

```
<Protocol  name ="Contract  Net" domain ="Negotiation  " version ="FIPA  00029">
   <Role  name ="Initiator  "/>
   <Role  name ="Participant  "/>
   <Message  ACT=" cfp " sender ="Initiator  " receiver  ="Participant  "/>
   <Message  ACT=" refuse  " sender ="Participant  " receiver  =" Initiat  or "/>
   <Message  ACT=" propose " sender =" Participant  " receiver  ="Initia  tor "/>
   <Message  ACT=" reject  -proposal " sender =" Initiator  " receiver  ="Participant  "/>
   <Message  ACT=" accept  -proposal " sender =" Initiator  " receiver  ="Participant  "/>
   <Message  ACT=" failure  " sender ="Participant  " receiver  =" Initia  tor "/>
   <Message  ACT=" inform  -done " sender =" Participant  " receiver  =" Initiator  "/>
   <Message  ACT=" inform  -result " sender ="Participant  " receiver  =" Initiator  "/>
   <State  name =" Initial " type ="initial "/>
   <State  name =" Announced  "/>
   <State  name =" Refused  " type ="final"/>
   <State  name =" Proposed  " type ="intermediary  "/>
   <State  name =" Rejected  " type ="final"/>
   <State  name =" Awarded  " type =" intermediary  "/>
   <State  name =" Failed  " type ="final"/>
   <State  name ="Finished  " type ="final"/>
   <Transition  State =" Initial " ActionType  ="XOR">
        <Action  ACT=" cfp " NextState  ="Announced  "/>
   </ Transition >
   <Transition  State ="Announced  " ActionType  ="XOR">
        <Action  ACT=" refuse " NextState  ="Refused  "/>
        <Action  ACT=" propose " NextState  =" Proposed  "/>
   </ Transition >
   <Transition  State =" Proposed  " ActionType  ="XOR">
        <Action  ACT=" reject -proposal " NextState  ="Rejected  "/>
        <Action  ACT=" accept -proposal " NextState  ="Awarded  "/>
    </ Transition >
    <Transition  State =" Awarded  " ActionType  ="XOR">
        <Action  ACT=" failure  " NextState  =" Failed  "/>
        <Action  ACT=" inform -done " NextState  =" Finished  "/>
        <Action  ACT=" inform -result " NextState  =" Finished  "/>
     </Transition  >
</ Protocol >
```

Figura 6: Descrição XML da FIPA Contract Net

Neste exemplo, será demonstrada uma parte de uma Conversa Contract Net para se ilustrar o funcionamento do ProtocolBox.

6.1 Instanciar conversa (agente iniciador)

Para se iniciar uma nova conversa, o agente iniciador deve acessar o método ***initiateConversation***, e passar como parâmetros o arquivo XML e os agentes com quem ele deseja efetuar a conversa. Feito isso, é instanciado um objeto **Conversation** (do tipo ***initiator***) no ProtocolBox. Este objeto armazena as informações como ***id*** da conversa, protocolo utilizado, além de informações sobre a interação com cada um dos participantes.
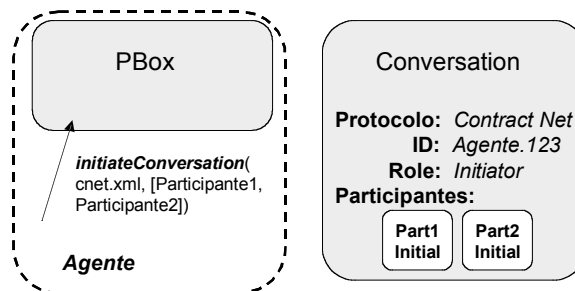
Figura 7: Método initiateConversation e objeto Conversation do tipo Initiator

6.2 Verificar possibilidades e enviar mensagem

Para verificar quais as ações que um agente pode realizar, existe um método chamado ***getPossibilities*** (Conversation). Este método retorna todas as ações que o agente pode realizar nesta conversa. Seguindo o exemplo da Contract Net, o método avisa ao agente que ele pode enviar um ***cfp*** (anúncio de tarefas) para os agentes Participante1 e Participante2 (figura 8.a)
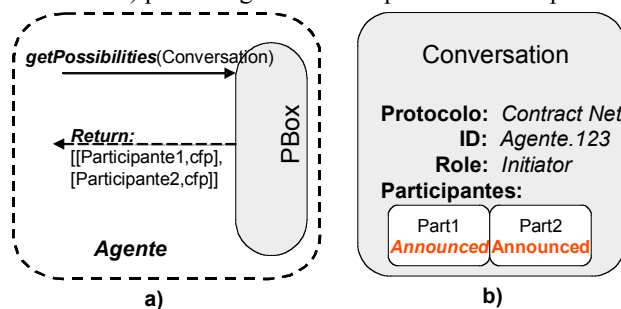
Figura 8: a) método ***getPossibilities*** b) Objeto conversation após o envio de mensagens para os Participantes 1 e 2 (observe alteração no estado das interações)

O agente então ***resolve*** enviar as mensagens ***cfp*** para os participantes 1 e 2. Para tanto, existe o método ***sendMessage*** que será chamado passando como parâmetros a performativa (no caso ***cfp***) e os agentes que irão receber as mensagens.

6.3 Recebimento de mensagens e instanciação de conversa participante

Os agentes ***participantes*** não possuem conhecimento de que estão fazendo parte de uma conversa até o momento em que chega a primeira mensagem. Quando uma mensagem referente a uma nova conversa chega no ProtocolBox, ele é responsável por instanciar um novo objeto **Conversation** (agora do tipo ***participant***) e avisar a chegada desta mensagem para o agente (através de ***handlers*** adequados).
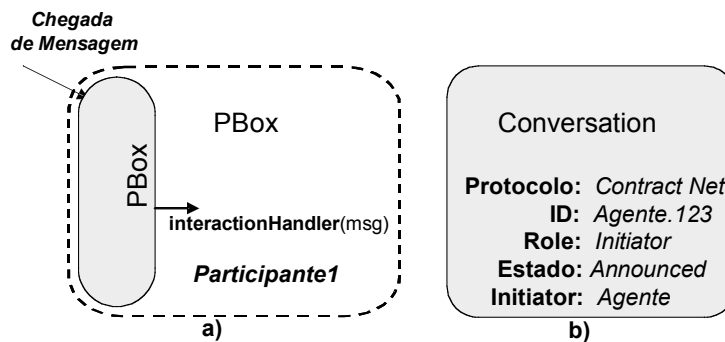
Figura 9: a) chegada de uma mensagem no ProtocolBox b) Objeto Conversation do tipo Participant instanciado após a chegada da mensagem

6.4 Outras funcionalidades do ProtocolBox

Este exemplo, que utiliza parte de uma conversa do tipo Contract Net, teve como objetivo ilustrar as funcionalidades básicas dos ProtocolBox. É importante lembrar que o este componente oferece outras funcionalidades que não foram descritas, como: controle de time-out, detecção de ações inválidas, gerenciamento e finalização das conversas.

7 Conclusões

Um dos fatores mais importantes no desenvolvimento de um Sistema Multiagentes se refere ao projeto da comunicação e interação entre os agentes. Para que a comunicação entre os agentes seja eficiente, é necessário que ela se enquadre a certas regras e mecanismos de interação entre agentes de modo a garantir objetividade nesta comunicação. Estes mecanismos de interação são fornecidos pelos chamados protocolos de comunicação ou protocolos de interação entre agentes

A implementação de novos componentes no ambiente SACI, permitindo que este passe a reconhecer e gerenciar os protocolos de comunicação, possibilita a reutilização em aplicações particulares de protocolos de interação genéricos, facilitando o processo de implementação de Sistemas Multiagentes.

Além disso, a utilização do ProtocolBox visa garantir, de uma maneira transparente ao programador, que as regras definidas nos protocolos de interação sejam seguidas corretamente. Desta forma, o programador não necessita se preocupar com detalhes referentes à implementação das interações, como ficar testando se uma mensagem recebida é válida.

Neste contexto, a utilização do ambiente SACI se torna cada vez mais atrativa para o desenvolvimento de Sistemas Multiagentes. Assim, o ambiente pode vir a auxiliar o desenvolvimento de outros trabalhos acadêmicos, o que pode inclusive vir a despertar o interesse de outros pesquisadores em se aperfeiçoar esta ferramenta.

Agradecimentos

Este trabalho foi desenvolvido durante um programa de Iniciação Científica do aluno Issao Hirata, que foi financiado pelo programa PIBIC do CNPq. Jaime Simão Sichman é parcialmente financiado pelo CNPq, proc. 482019/2004-2 e 304605/2004-2.

Referências

Foundation for Intelligent Physical Agents. FIPA Contract Net Interaction Protocol Specification, Geneva, 2002. Disponível em <http://www.fipa.org/specs/fipa00029>. Acesso em 10/09/2004

Foundation for Intelligent Physical Agents. FIPA ACL Message Structure Specification, Geneva, 2002. Disponível em <http://www.fipa.org/specs/fipa00061>. Acesso em 17/12/2004

HÜBNER, J. F. , SICHMAN, J. S. SACI: Uma ferramenta para implementação e monitoração da comunicação entre agentes. In: M. C. Monard e J. S. Sichman eds. **Anais do IBERAMIA/SBIA 2000**, Open Discussion Track, pages 47-56, Atibaia, SP, Brasil, 2000.

LABROU, Y. FININ, T., PENG, Y. Agent communication languages: the current landscape. **IEEE Intelligent Systems**, v. 14, n. 2, p. 45-52, 1999.

ODELL, J. J., PARUNAK, V. D. H., BAUER, B., Representing Agent Interaction Protocols in UML. In: **Agent-Oriented Software Engineering**, Ciancarini, P. and Wooldridge, M., Eds., Springer, pages. 121- 140, Berlin, 2001.

POPULAIRE P., BOISSER O., SICHMAN J. S. Description et Implementation de Protocoles de Communication en Univers Multi-Agents. **Premières journées francophones IAD & SMA**, Tolouse, 1993

SACI. Simple Agent Communication Infrastucture. Disponível em <http://www.lti.pcs.usp.br/saci>. Acesso em 05/10/2004

SEARLE, J.R. **Speech Acts: an Essay in the Philosophy of Language**. Cambridge U. Pres, 1970.

SMITH, R. G. The Contract Net Protocol: high-level communication and control in a distributed problem solver. **IEEE Transactions on Computers**, v.29, n.12, p. 1104-1113, 1980.

WOOLDRIDGE, M., **An Introduction to MultiAgent Systems**. John Wiley & Sons, UK. 2002.

Aplicação de Autômatos Finitos Nebulosos no Reconhecimento Aproximado de Cadeias

Alexandre Maciel (PCS/EPUSP)
alexandre.maciel@poli.usp.br

Marco Túlio Carvalho de Andrade (PCS/EPUSP)
marco.andrade@poli.usp.br

Resumo. Esse artigo discute as características e as aplicações do reconhecimento aproximado de cadeias, e propõe uma abordagem alternativa para a sua resolução: o uso dos *autômatos finitos nebulosos*. Os autômatos finitos nebulosos combinam a facilidade de modelagem proporcionada pelos autômatos finitos não-determinísticos com o sofisticado aparato matemático para o tratamento de informações sujeitas a erros, característico da teoria nebulosa: esse artigo discute brevemente o seu funcionamento e suas vantagens.

Palavras-chave: Fundamentos da Computação, Lógica Nebulosa, Reconhecimento Aproximado de Cadeias.

1 Introdução

Com a evolução da computação nos últimos anos, o crescimento vertiginoso da quantidade de dados armazenados e processados criou novos e interessantes problemas: as técnicas convencionais de indexação, procura e processamento das informações só funcionam com dados rigorosamente livres de erros. E, enquanto retificar manualmente um conjunto relativamente pequeno de dados (algo como 10 quilobytes, por exemplo) é uma tarefa trivial, é impraticável utilizar a mesma solução para quantidades de dados da ordem de gigabytes ou terabytes.

Junto com problemas clássicos de computação, surgem também problemas novos que, inerentemente, possuem uma grande quantidade de dados com uma margem de erro não-desprezível para serem processados. Tais problemas, que não poderiam ser tratados com a tecnologia disponível há apenas alguns anos atrás, também necessitam de soluções não-convencionais para serem resolvidos computacionalmente com eficiência.

Esse artigo tem o objetivo de apresentar uma nova técnica para tratar alguns desses problemas. Essa técnica permite fazer *reconhecimento aproximado de cadeias*, também conhecido como *reconhecimento de cadeias com tolerância a erros*, utilizando *autômatos finitos nebulosos*. O objetivo dessa técnica é casar um padrão com um texto dado, considerando que um dos dois (ou mesmo os dois) sofreram algum tipo de corrupção indesejada. Mais do que isso, a técnica apresentada permite quantificar o grau dessa corrupção e ajustar a tolerância utilizada através de parâmetros possibilísticos.

Existem diversas áreas de aplicação potencial para esse tipo de técnica. No entanto, três grandes áreas se destacam: recuperação dos sinais originais após a transmissão deste sinal através de um canal ruidoso, procura por seqüências de DNA após mutações genéticas e busca por termos em textos com erros de digitação ou ortografia (NAVARRO, 2001).

Esse artigo está organizado da seguinte maneira: a Seção 2 deste documento aborda algumas aplicações de reconhecimento aproximado de cadeias. A Seção 3 apresenta mais de informações sobre *reconhecimento aproximado de cadeias* e *teoria nebulosa*.

Na Seção 4 é apresentado o modelo em si, com informações sobre o seu funcionamento, exemplos e considerações. Finalmente, a Seção 5 conclui o artigo e discute os resultados obtidos.

2 Aplicações do Reconhecimento Aproximado de Cadeias

Como foi antecipado na introdução, o reconhecimento aproximado de cadeias é utilizado quando se necessita fazer uma busca por um padrão em uma cadeia sujeita a diversos erros indesejáveis. Podemos dividir as aplicações do reconhecimento aproximado de cadeias em três grandes áreas: biologia computacional, processamento de sinais e reconhecimento de textos.

2.1 Biologia Computacional

O Projeto Genoma e as demais pesquisas recentes no campo da engenharia genética trouxeram a tona problemas interessantes e novos, inclusive problemas para a ciência da computação. O DNA é responsável pela codificação integral de todas as características de um organismo vivo: estudar e decifrar o seu significado poderá, no futuro, garantir o acesso à tecnologia médica e biológica inimaginável atualmente.

A despeito de sua importância, o DNA nada mais é do que um longuíssimo encadeamento de 4 bases nitrogenadas distintas, a saber: Adenina, Citosina, Guanina e Timidina. Alguns consideram também uma quinta base nitrogenada, a Uracila, gerada a partir de uma mutação indesejável da Citosina.

Apesar de ser uma aplicação de reconhecimento de cadeias, o conjunto de aplicações onde o reconhecimento estrito de cadeias pode ser utilizado é extremamente restrito: diversos fatores tais como erros inerentes ao processo de coleta de dados, mutações genéticas menos significativas, alterações evolucionárias e outros fatores fazem com que a busca exata por seqüências de bases nitrogenadas raramente retornem algum resultado útil.

O reconhecimento aproximado de cadeias, por outro lado, oferece resultados muito mais satisfatórios para os pesquisadores. Em anos de Projeto Genoma e outras pesquisas, os geneticistas adquiriram um extenso conhecimento sobre os erros observados nas seqüências de DNA que acontecem com freqüência: se o mecanismo utilizado no reconhecimento aproximado de cadeias for capaz de levar em conta esse conhecimento e atribuir um peso menor a esses erros ao quantificar a diferença entre a seqüência genética desejada e a observada, melhor ainda. Além disso, a quantificação das diferenças entre duas seqüências de genes é bastante útil para a reconstrução de árvores evolucionárias (árvores filogenéticas).

2.2 Processamento de Sinais

Uma outra área onde o reconhecimento aproximado é bastante utilizado é no processamento de sinais, sobretudo na correção de erros e no reconhecimento de voz.

Uma vez que toda informação transmitida através de um meio físico está inexoravelmente sujeita a erros, podemos recorrer à Teoria da Informação (SHANNON, 1948) para obter as probabilidades de ocorrência de erros e usar algum mecanismo para recuperar a informação contida na mensagem original (na realidade, processar o sinal recebido e obter uma mensagem recebida mais próxima da mensagem transmitida, uma vez que a própria Teoria da Informação determina a impossibilidade da recuperação integral da mensagem transmitida).

Entretanto, falar em "reconhecimento aproximado de cadeias" em correção de erros não é o mais apropriado: geralmente não se deseja encontrar um padrão, apenas eliminar os erros de uma mensagem recebida (seja qual for o critério utilizado pelo mecanismo para definir 'erro'). Exatamente por isso, as contribuições do reconhecimento de erros à busca aproximada são bastante incipientes, no entanto, a área é responsável por uma das mais importantes e utilizadas métricas de similaridade entre cadeias, a *distância de Levenshtein*.

O reconhecimento de voz, por outro lado, é uma aplicação bastante característica do reconhecimento aproximado de cadeias. Consiste, basicamente, em determinar uma cadeia de caracteres (ou

seja, um texto), a partir de um sinal de áudio. É praticamente impossível obter o reconhecimento estrito perfeito: existem diversos deslocamentos temporais no sinal, diferenças de timbre e pronúncia mesmo por dois sinais de áudio gerados pela mesma pessoa e sons ambientes que interferem no sinal, por exemplo.

Dessa forma, mesmo distinguir uma palavra de um repertório extremamente limitado (algo como 20 ou 30 palavras, por exemplo), é uma tarefa que não pode ser feita sem a utilização do reconhecimento aproximado.

O crescimento do uso de interfaces homem-máquina não-escritas pelos sistemas computacionais mais modernos abre grandes possibilidades para o uso de mecanismos de reconhecimento aproximado de cadeias. Bancos de dados multimídia, por exemplo, armazenam imagens, sons e vídeos que, eventualmente, precisaram passar por buscas aproximadas, não-estritas (em raríssimas situações seria útil, por exemplo, fazer a busca estrita de um padrão de imagem num banco de dados de imagens); formas de interação com computadores através de comandos de voz e gestos (VOLLET, 2001)): todas essas aplicações demandam mecanismos de reconhecimento aproximado.

Além disso, a disseminação do uso de sistemas de comunicação sem fio (*wireless*) demanda por mecanismos de correção de erros cada vez mais sofisticados: o ar por si só é um péssimo meio de transmissão de sinais digitais, a saturação cada vez maior das bandas de transmissão disponíveis torna o problema ainda pior. Dessa forma, há um enorme potencial do uso de mecanismos de reconhecimento aproximado de cadeias e/ou correção de erros na área de processamento de sinais para os próximos anos.

2.3 Recuperação de Texto

A recuperação de texto é uma das mais antigas e disseminadas aplicações onde o reconhecimento aproximado de cadeias é utilizado. Existem diversas áreas e esse problema aparece, mas a *recupera- ção de informações* é uma das mais críticas.

Recuperação de informações nada mais é do que procurar por determinado padrão em uma quantidade relativamente grande de texto; o reconhecimento de cadeias é uma de suas ferramentas mais básicas.

Embora pequenas quantidades de texto possam ser corrigidas manualmente e servir como base para a recuperação de informações utilizando ferramentas de reconhecimento estrito de cadeias; a recuperação de informações só se torna atraente com uma quantidade substancialmente maior de texto, como a World Wide Web, com uma quantidade de texto da ordem de dezenas de gigabytes ou maior.

Utilizando um mecanismo de reconhecimento estrito de cadeias pode fazer com que um termo inserido com algum erro na base de dados jamais possa ser recuperado. Por exemplo, textos inseridos através de OCR (Optical Character Recognition - reconhecimento óptico de caracteres) possuem uma taxa de erro entre 7% e 16%. Erros de digitação atingem entre 1% e 3,2% dos textos, enquanto erros ortográficos respondem por taxas de erro entre 1,5% e 2,5%. A despeito da freqüência substancial de erros que ocorrem ao inserir o texto na base do sistema de recuperação de informações - seja manualmente ou não - 80% desses erros podem ser corrigidos através de uma única operação básica de reconhecimento aproximado de cadeias (inserção, exclusão, substituição e transposição - maiores detalhes na Seção 3. Todas essas estatísticas foram retiradas do trabalho de (NAVARRO, 2001).

Levando-se em conta a crescente necessidade de fazer a recuperação de informações em bases de dados multi-linguais (como a World Wide Web, mas também um banco de dados corporativo de uma empresa multinacional) e alimentadas com informações geradas ao longo de anos ou décadas, a utilização de aplicações que implementam algum algoritmo de reconhecimento aproximado de

cadeias torna-se cada vez mais necessária, na realidade, poucas são as aplicações de recuperação de informações que não utilizam. Dessa forma, podemos encontrar informações a partir de uma palavra ou nome estrangeiro grafado erroneamente (por exemplo, 'Greenhalg' ao invés de 'Greenhalgh') ou encontrar uma informação escrita em forma arcaica do idioma utilizado através um padrão em na forma moderna (como procurar por 'farmácia' e encontrar 'pharmácia').

3 Conceitos

Essa seção introduz alguns conceitos básicos, imprescindíveis para a compreensão do modelo proposto nesse artigo. Ele não visa, no entanto, ser um tutorial completo sobre teoria nebulosa ou reconhecimento aproximado de cadeias. Esses temas são abordados de maneira mais profunda em (NAVARRO, 2001) (reconhecimento aproximado de cadeias) e em (NGUYEN; WALKER, 2000) (teoria nebulosa).

3.1 Reconhecimento Aproximado de Cadeias

Na Introdução do artigo, o problema do reconhecimento de cadeias foi definido de maneira informal. Entretanto, é necessário definir o problema de maneira mais formal, segue abaixo a definição do problema.

Definição 1 *Considere o seguinte:*

Seja Σ *um alfabeto finito com* $n = |\Sigma|$ *símbolos distintos;*
Seja ω *o padrão procurado, tal que* $\omega = \{x_1 x_2 ... x_m\}$ *e* $m = |\omega|$*;*
Seja α *o texto onde a busca é feita, tal que* $\alpha = \{y_1 y_2 ... y_o\}$ *e* $o = |\alpha|$*;*
Seja $k \in \Re$ *o erro máximo tolerado;*
Seja $d : \Sigma \times \Sigma \rightarrow \Re$ *a função de distância;*
Seja $d' : \Sigma \times \Sigma \rightarrow \Re$ *a função de similaridade.*

Dessa forma, o problema do reconhecimento aproximado de cadeias é definido da seguinte maneira: dado ω*,* α*,* k *e* $d(\cdot)$*, quais os valores de* j*, tal que exista um* i*, de maneira que* $d(\omega, (y_i ... y_j) \leq k$ *(distância) ou* $d(\omega, (y_i ... y_j)) \geq k$ *(similaridade).*

Podemos definir a função $d(\beta, \omega)$ (ou a função $d'(\beta, \omega)$) como o menor custo possível associado às *operações* necessárias para transformar β em ω. As operações, por sua vez, nada mais são do que um conjunto finito de regras no formato $\kappa(a, b) = c$, onde $a, b \in \Sigma^*$, $a \neq b$ e $c \in \Re^+$. O custo total da função $d(\cdot)$ é contabilizado a partir dos custos individuais das operações (geralmente, mas não necessariamente, através da soma).

A despeito do fato de que alguns trabalhos definem outras operações específicas para algumas aplicações em particular, podemos considerar as seguintes operações básicas como todas as possíveis:

- *Inserção*: Insere um caractere ausente na cadeia $y_i ... y_j$, de maneira que a operação $\kappa(\varepsilon, a)$ a transforma em $y_i ... a ... y_j$, onde $a \in \Sigma$.
- *Exclusão*: Excluí um caractere presente na cadeia $y_i ... y_{k-1} y_k y_{k+1} ... y_j$, de maneira que a operação $\kappa(y_k, \varepsilon)$ a transforma em $y_i ... y_{k-1} y_{k+1} ... y_j$.
- *Substituição*: Substitui um caractere presente na cadeia $y_i ... y_{k-1} y_k y_{k+1} ... y_j$ por outro caractere, de maneira que a operação $\kappa(y_k, a)$ a transforma em $y_i ... y_{k-1} a y_{k+1} ... y_j$, onde $a \in \Sigma$ e $a \neq y_k$.

- *Transposição*: Troca a posição de dois símbolos consecutivos presentes na cadeia $y_i...y_{k-1}y_ky_{k+1}...y_j$, de maneira que a operação $\kappa(y_{k+1}, y_{k+1}y_k)$ a transforma em $y_i...y_{k-1}y_{k+1}y_k...y_j$, onde $y_k \neq y_{k+1}$.

Nem todos os mecanismos de reconhecimento aproximado e métricas de distância/similaridade trabalham com todas as operações acima descritas. De fato, é possível simular uma operação de substituição através de uma inserção seguida de uma exclusão; assim como é possível simular uma operação de transposição através de duas operações de substituição consecutivas. Obviamente, uma vez que são necessárias duas operações (ou quatro, quando simulando uma transposição apenas com inclusões e exclusões) para simular uma única operação, o custo associado é diferente e, dependendo da métrica utilizada, os resultados também podem ser diferentes. Também é importante ressaltar que, embora os erros de transposição sejam muito freqüentes (NAVARRO, 2001), existem poucos mecanismos de reconhecimento aproximado de cadeias capazes de lidar com transposições.

Por exemplo, a *distância de Levenshtein* (também conhecida como *distância de edição*) é uma função de distância que utiliza apenas as operações de inserção, exclusão e substituição; calculando a distância entre duas cadeias a partir da soma algébrica dos custos individuais das operações, que sempre são iguais a 1. Esse trabalho utiliza uma métrica conhecida como *distância de Levenshtein generalizada*: também utiliza apenas as operações de inserção, exclusão e substituição, embora cada operação em cada contexto possa ter um custo associado diferente e os custos individuais das operações são agregados a partir de uma operação arbitrária, não necessariamente a soma algébrica (essa operação, na verdade, determina se a distância de Levenshtein generalizada é uma função de distância ou de similaridade).

3.2 Teoria Nebulosa

Enquanto a computação clássica trabalha manipulando valores bem definidos em modelos matemáticos concretos e categóricos, a razão humana é capaz de trabalhar com valores dotados de uma imprecisão relativamente alta em modelos aproximados com uma eficiência surpreendente. Desta forma faz-se necessária a adoção de uma nova abordagem quando se quer resolver, computacionalmente, problemas facilmente resolvidos pelo ser humano. Uma abordagem que não utilize a precisão, o rigor e o formalismo matemático, mas que seja tolerante a falhas e verdades parciais (ZADEH, 1973).

A *Teoria Nebulosa* permite o tratamento desta classe de problemas explorando o conceito de *conjuntos nebulosos*. Enquanto a relação entre um conjunto clássico e um elemento do conjunto universo só admite duas possibilidades: pertence e não-pertence; os conjuntos nebulosos permitem quantificar de maneira mais granular a relação entre conjuntos e elementos. De fato, uma função associa o *grau de pertinência* de um elemento a um conjunto através de um número real no intervalo $[0, 1]$.

Na realidade, os conjuntos clássicos são um caso específico dos conjuntos nebulosos: um grau de pertinência 0 significa "não-pertence", enquanto um grau de pertinência 1 significa "pertence".

Por exemplo, considere o conjunto nebuloso $\tilde{L} = \{$ conjunto dos pesos leves $\}$. Poderíamos dizer que 100g pertence ao conjunto, enquanto 500kg não. Mas e quanto a 200g ou 1kg? Poderíamos representar essa informação da seguinte maneira: $\tilde{L} = \{$100g$/1$; 200g$/0,9$; 1kg$/0,4$; 500kg$/0\}$. Na verdade, não precisamos representar exaustivamente todos os elementos e seus graus de pertinência: podemos omitir os elementos cujo grau de pertinência é igual a zero (no caso, 500kg - sempre se assume que os elementos não-representados têm o valor de pertinência igual a 0) ou podemos representar o conjunto nebuloso pela função de pertinência característica.

Dessa forma, os conjuntos nebulosos são ferramentas simples e poderosas para a representação de informações sujeitas a erros e a distorções: sobretudo a representação de informações qualitativas (no exemplo, o grau de "leveza" de determinado peso).

4 Reconhecimento Aproximado de Cadeias com Autômatos Finitos Nebulosos

Uma vez que os conceitos fundamentais foram assimilados, podemos agora fazer o aprofundamento do tema e verificar como os autômatos finitos nebulosos funcionam e como podem ser utilizados no reconhecimento aproximado de cadeias.

4.1 Autômato Finito Nebuloso

O *autômato finito nebuloso* é um modelo derivado do autômato finito convencional, onde alguns conjuntos que servem de definição a este são substituídos por conjuntos nebulosos. Isso permite que o autômato finito nebuloso faça o reconhecimento de diversas cadeias diferentes; atribuindo diferentes graus de pertinência para cada uma delas. De fato, falar em "aceitação" de uma cadeia por um autômato finito nebuloso é um equivoco: estritamente ele é capaz de reconhecer qualquer cadeia criada a partir do seu alfabeto, ainda que algumas cadeias tenham grau de pertinência 0.

Segue abaixo a definição do autômato finito nebuloso, passo a passo.

Definição 2 *Um Autômato Finito Nebuloso é uma quíntupla* $\tilde{M} = (Q, \Sigma, \mu, \tilde{S}, \tilde{F})$*, onde:*

> Q *é o conjunto (clássico) finito de estados,*
> Σ *é o conjunto de símbolos (alfabeto) da máquina,*
> μ *é a relação de transição nebulosa* $Q \times \{\Sigma \cup \varepsilon\} \times Q \rightarrow [0, 1]$,
> $\tilde{S} \subseteq Q$ *é o conjunto nebuloso de estados iniciais,*
> $\tilde{F} \subseteq Q$ *é o conjunto nebuloso de estados finais.*

Dessa forma, cada transição tem um valor próprio de pertinência que, associado ao valor de pertinência dos elementos dos conjuntos nebulosos dos estados iniciais e finais, determina o grau de pertinência da cadeia para determinado autômato.

As definições de como as transições, em geral, são computadas segue adiante:

Definição 3 *Seja* $F(Q)$ *o conjunto de todos os conjuntos nebulosos possíveis sobre Q. Dessa forma, a relação* $\mu^* : F(Q) \times \Sigma^* \rightarrow F(Q)$ *seria definida como se segue:*

$$\mu^*(\tilde{P}, \omega) = \begin{cases} \mu^\varepsilon(\tilde{P}),\ \tilde{P} \in F(Q),\ \textit{se}\ \omega = \varepsilon; \\ \mu^\varepsilon(\hat{\mu}(\mu^*(\tilde{P}, \omega'), a)),\ \omega = \omega' a,\ \omega' \in \Sigma^*,\ a \in \Sigma \tilde{P} \in F(Q),\ \textit{caso contrário.} \end{cases}$$

A relação $\hat{\mu}$ é responsável por fazer as transições que demandam que um caractere seja consumido, enquanto a relação μ^ε é responsável por fazer as transições vazias. O funcionamento do autômato consiste em chamá-las alternadamente até que toda a cadeia seja consumida.

A definição da relação $\hat{\mu}$ é a mais simples das duas, e está logo abaixo:

Definição 4 *A relação $\hat{\mu} : F(Q) \times \Sigma \to F(Q)$ é definida como:*

$$\hat{\mu}(\tilde{P}, x) = \{(p, \mu) \mid \mu = \oplus_{p \in Q}(\mu(q, x, p) \otimes \mu_{\tilde{P}}(q)), p \in Q\}, \text{ onde } \tilde{P} \in F(Q), x \in \Sigma$$

O símbolo $\otimes$ representa uma *norma triangular*, que é o equivalente nebuloso para a intersecção entre conjuntos (ou o operador booleano “E”). Ele é utilizado para computar o valor de pertinência atual dos estados a partir do valor de pertinência dos estados anterior e do valor de pertinência das transições que estão sendo realizadas. Por sua vez, o símbolo $\oplus$ representa uma *conorma triangular*, que é o equivalente nebuloso para a união entre conjuntos (ou o operador booleano “OU”). Ele é utilizado para computar o valor de pertinência de um estado, quando existem duas ou mais transições que levam a esse estado.

Já a definição da relação μ^{ε} é descrita adiante:

Definição 5 *Seja $F(Q)$ o conjunto de todos os possíveis conjuntos nebulosos sobre Q, e seja a relação $\mu^{\varepsilon} : F(Q) \to F(Q)$. Para calcular μ^{ε}, é necessário fazer algumas definições adicionais.*

Seja $\langle q_{k_0} q_{k_1} ... q_{k_l} \rangle$ uma seqüência de estados de Q finita e maior do que zero, de maneira que $\forall i, j, 0 \leq i, j \leq l, i \neq j$ então $q_{k_i} \neq q_{k_j}$. Em outras palavras, significa que a seqüência de estados $\langle q_{k_0} q_{k_1} ... q_{k_l} \rangle$ é acíclica, ou seja, a seqüência não percorre duas vezes o mesmo estado.

Seja $\tilde{E}(q)$ um conjunto nebuloso de estados tal que $\tilde{E}(q) \in F(Q)$. Esse conjunto determina representa os estados e seus respectivos valores de pertinência que podem ser atingidos a partir do estado ‘q’ usando apenas transições vazias, e é definido como:

$$\tilde{E}(q) = \{(p, \mu) \mid \mu = \oplus_{\langle q_{k_0} ... q_{k_l} \rangle, q_{k_0} = q, q_{k_l} = p}(\otimes_{0 < i \leq l}\, \mu(q_{k_{i-1}}, \varepsilon, q_{k_i})), p \in Q\}.$$

Dessa forma, a relação μ^{ε} ficaria:

$$\mu^{\varepsilon}(\tilde{P}) = \{(p, \mu) \mid \mu = \mu_{\tilde{P}}(p) \oplus (\oplus_{q \in Q}(\mu_{\tilde{E}(q)}(p) \otimes \mu_{\tilde{P}(q)})), p \in Q\}, \text{ onde } \tilde{P} \in F(Q).$$

Sendo que os símbolos $\otimes$ e $\oplus$ tem o mesmo significado da Definição 4, com uma pequena ressalva. Uma vez que é necessária a manutenção da propriedade da idempotência, já que $\mid \varepsilon \mid = 0$ e $\varepsilon \equiv \varepsilon\varepsilon \equiv \varepsilon\varepsilon\varepsilon \equiv ...$, o único operador de conorma triangular aceitável na relação μ^{ε} é a *união nebulosa (máximo)*. O raciocínio para entender esse fato é um pouco extenso para ser desenvolvido nesse artigo, mas consultas a (LEE, 1990b) e (VOXMAN; GOETSHCHEL, 1983), assim como (NGUYEN; WALKER, 2000) são suficientes para entendê-lo. As normas triangulares assim como as conormas triangulares das outras relações podem ser escolhidas sem essa restrição.

Podemos definir o conjunto nebuloso de cadeias determinado por um autômato finito nebuloso (o equivalente para a linguagem aceita por um autômato finito clássico) da seguinte maneira:

Definição 6 *Seja ω uma cadeia qualquer, de maneira que $\omega \in \Sigma^*$ e $\tilde{M}$ o autômato finito nebuloso que determina o valor de pertinência dessa cadeia. Portanto:*

$$\tilde{L}(\tilde{M}) = \{(\omega, \mu_{\tilde{M}}(\omega)) \,|\, \omega \in \Sigma^*\}$$

Por fim, a função $\mu_{\tilde{M}}(\omega)$, responsável por calcular os valores de pertinência para as cadeias do autômato finito nebuloso, é dada por:

$$\mu_{\tilde{M}}(\omega) = \oplus_{q \in Q}(\mu_{\mu^*(\tilde{S}, \omega)}(q) \otimes \mu_{\tilde{F}}(q))$$

4.2 Reconhecimento Aproximado de Cadeias utilizando Autômatos Finitos Nebulosos

Vamos ver agora como utilizar o modelo do autômato finito nebuloso apresentado anteriormente para fazer o reconhecimento aproximado de cadeias. Entretanto, antes vamos fazer algumas observações.

O emprego de autômatos finitos nebulosos para o reconhecimento aproximado de cadeias permite a liberdade de escolha dos operadores de norma e conorma triangulares (exceto para a conorma triangular utilizada em μ^{ε}, como foi visto na Definição 5). Esse trabalho utiliza como operador de norma triangular o *produto algébrico* e como operador de conorma triangular a *união nebulosa (máximo)*. A métrica utilizada é a da *distância de Levenshtein generalizada*, que determina a similaridade, e não a diferença, entre duas cadeias. Entretanto, outras combinações de operadores podem ser utilizadas, criando métricas diferentes e gerando resultados distintos, possivelmente úteis para aplicações específicas.

Sendo assim, o algoritmo que cria um autômato finito nebuloso capaz de fazer o reconhecimento aproximado de uma cadeia é o seguinte:

Definição 7 *Seja* $\omega = \{x_1 x_2 ... x_m\}$ *uma cadeia de símbolos sobre* Σ*, de maneira que* $m = \mid \omega \mid$*; e seja* $\tilde{M}_\omega = (Q, \Sigma, \mu, \tilde{S}, \tilde{F})$ *o autômato finito nebuloso capaz de fazer o reconhecimento aproximado dessa cadeia de acordo com a distância de Levinshtein generalizada. Portanto, temos:*

$Q = \{q_0, q_1, ..., q_m\}$ *é o conjunto clássico de estados do autômato finito nebuloso,*
Σ *é o conjunto finito de símbolos (alfabeto) do autômato finito nebuloso,*
μ *é a relação de transição* $Q \times \{\Sigma \cup \varepsilon\} \times Q \rightarrow [0, 1]$*, descrita separadamente.*
$\tilde{S} = \{q_0/1\}$ *é o conjunto nebuloso de estados iniciais,*
$\tilde{F} = \{q_m/1\}$ *é o conjunto nebuloso de estados finais.*

A função de transição é responsável pela representação das operações de inclusão, exclusão e substituição no autômato nebuloso. Seja $0 < i \leq \mid \omega \mid$, $j \in (\Sigma - \{x_i\})$ *e* $k \in \Sigma$*. Dessa maneira, temos que a função de transição* μ *é dada por:*

- $\forall i$: $\mu(q_{i-1}, x_i, q_i) = 1$ *(aceitação);*
- $\forall i, \forall j$: $\mu(q_{i-1}, j, q_i) = \kappa(j, x_i)$ *(substituição);*
- $\forall i$: $\mu(q_{i-1}, \varepsilon, q_i) = \kappa(\varepsilon, x_i)$ *(inserção);*
- $\forall i, \forall j$: $\mu(q_{i-1}, j, q_{i-1}) = \kappa(j, \varepsilon)$ *(exclusão I);*
- $\forall k$: $\mu(q_m, k, q_m) = \kappa(k, \varepsilon)$ *(exclusão II).*

Para ilustrar o funcionamento do autômato finito nebuloso, vamos a um exemplo simples.

Seja $\tilde{M}_1$ um autômato finito nebuloso capaz de fazer o reconhecimento da cadeia $\omega =$‘bau’. A especificação do autômato, bem como o seu funcionamento para a cadeia de entrada $\alpha =$‘aa’ é mostrada abaixo:

$\tilde{M}_1 = (Q, \Sigma, \mu, \tilde{S}, \tilde{F})$. Assim temos:
$Q = \{q_0, q_1, q_2, q_3\}$,
$\Sigma = \{a, b, ..., z\}$,
μ é definido como na Figura 1,
$\tilde{S} = \{q_0/1\}$,
$\tilde{F} = \{q_3/1\}$.
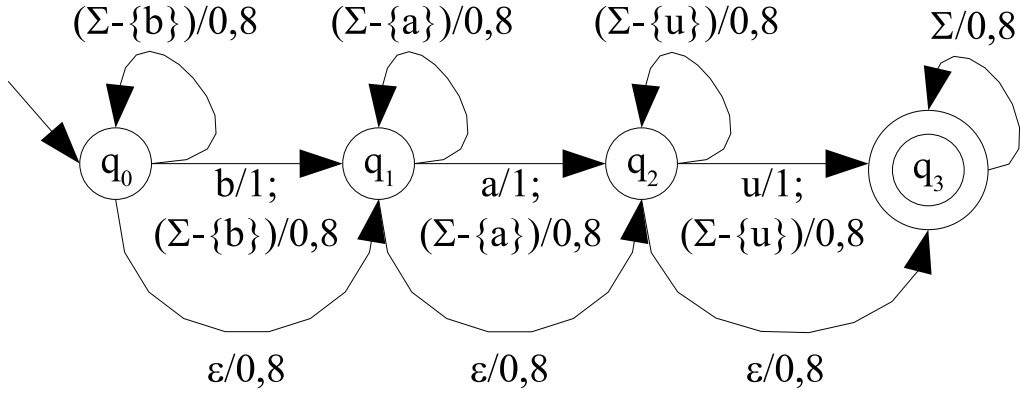
Figura 1: Autômato Finito Nebuloso $\tilde{M}_1$.

Uma vez que o conjunto nebuloso de estados finais possui apenas o estado q_3 com grau de pertinência diferente de zero, podemos considerar que:

$$\mu_{\tilde{M}_1}(aa) = (\mu_{\mu^*(\tilde{S},aa)}(q_3) \cdot 1) = \mu_{\mu^*(\tilde{S},aa)}(q_3)$$

Por outro lado, expandindo μ^* temos que:

$$\mu^*(\tilde{S}, aa) = \mu^\varepsilon(\hat{\mu}((\mu^*(\tilde{S}, a)), a)) = \mu^\varepsilon(\hat{\mu}((\mu^\varepsilon(\hat{\mu}((\mu^*(\tilde{S}, \varepsilon)), a))), a)) = \\ \mu^\varepsilon(\hat{\mu}((\mu^\varepsilon(\hat{\mu}(\mu^\varepsilon(\tilde{S})), a))), a))$$

A Tabela 1 ajuda a compreender cada etapa do processamento do autômato finito nebuloso.

	$\mu(q_0)$	$\mu(q_1)$	$\mu(q_2)$	$\mu(q_3)$
$\tilde{Q}_0 \equiv \tilde{S}$	1	0	0	0
$\tilde{Q}_1 \equiv (\mu^\varepsilon(\tilde{S}))$	1	$0,8$	$0,64$	$0,512$
$\tilde{Q}_2 \equiv \hat{\mu}(\mu^\varepsilon(\tilde{S}))$	$0,8$	$0,8$	$0,8$	$0,512$
$\tilde{Q}_3 \equiv \mu^\varepsilon(\hat{\mu}(\mu^\varepsilon(\tilde{S})))$	$0,8$	$0,8$	$0,8$	$0,64$
$\tilde{Q}_4 \equiv \hat{\mu}(\mu^\varepsilon(\hat{\mu}(\mu^\varepsilon(\tilde{S}))))$	$0,64$	$0,64$	$0,8$	$0,64$
$\tilde{Q}_5 \equiv \mu^\varepsilon(\hat{\mu}(\mu^\varepsilon(\hat{\mu}(\mu^\varepsilon(\tilde{S})))))$	$0,64$	$0,64$	$0,8$	$\mathbf{0,64}$

Tabela 1: Processamento da cadeia $\alpha =$‘aa’ pelo Autômato Finito Nebuloso $\tilde{M}_1$.

Dessa forma, o grau de pertinência da cadeia α no conjunto nebuloso definido pelo autômato $\tilde{M}_1$ é de 0,64. Observando o processamento, podemos notar também que $\kappa(a, b)$ e $\kappa(\varepsilon, u)$ são as operações mínimas necessárias para transformar α em ω.

Uma vez que a implementação do reconhecimento aproximado de cadeias utilizando autômatos finitos nebulosos foi compreendida, vamos ilustrar como é possível modelar o conhecimento sobre determinado problema nas funções de pertinência equivalentes aos pesos das operações de maneira simples e intuitiva utilizando o modelo proposto.

Considere o problema de busca por bases nitrogenadas em uma cadeia de DNA. Podemos imaginar uma cadeia de DNA como uma dupla fita helicoidal, cujas bases nitrogenadas sempre aparecem

em pares bem definidos: a Adenina com a Timidina e a Citosina com a Guanina. Uma quinta base nitrogenada, a Uracila, eventualmente surge, como fruto da conversão indesejável da Citosina.

Podemos definir o problema da busca por uma determinada seqüência de bases nitrogenadas num gene como o reconhecimento de cadeias. Como foi discutido na Subseção 2.1, o reconhecimento estrito de cadeias não produz resultados satisfatórios: é necessário empregar o reconhecimento aproximado de cadeias.

Sendo assim, considere o alfabeto $\Sigma_2 = \{A, C, G, T, U\}$ e o autômato finito nebuloso $\tilde{M}_2$ capaz de reconhecer a seqüência de bases 'AACT', como definido na Figura 2.
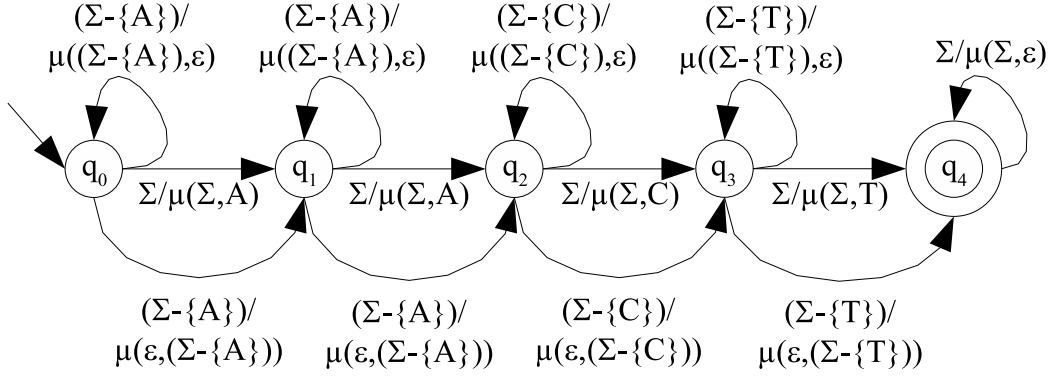
Figura 2: Autômato Finito Nebuloso $\tilde{M}_2$.

Uma vez que as bases nitrogenadas geralmente aparecem em duplas (Adenina-Timidina, Citosina-Guanina), é razoável assumir que encontrar uma cadeia "inversa" da cadeia procurada é um resultado relevante. Outra suposição razoável é imaginar que uma cadeia onde a Uracila está localizada em uma posição onde se esperava encontrar Citosina é uma resultado também relevante. A Tabela 2 contém os valores de pertinência para as funções de transição do autômato, mostrando como representar o conhecimento sobre o problema.

$\mu(A, A) = 1,0$	$\mu(A, C) = 0,3$	$\mu(A, G) = 0,3$	$\mu(A, T) = 0,7$	$\mu(A, \varepsilon) = 0,5$
$\mu(C, A) = 0,3$	$\mu(C, C) = 1,0$	$\mu(C, G) = 0,7$	$\mu(C, T) = 0,3$	$\mu(C, \varepsilon) = 0,5$
$\mu(G, A) = 0,3$	$\mu(G, C) = 0,7$	$\mu(G, G) = 1,0$	$\mu(G, T) = 0,3$	$\mu(G, \varepsilon) = 0,5$
$\mu(T, A) = 0,7$	$\mu(T, C) = 0,3$	$\mu(T, G) = 0,3$	$\mu(T, T) = 1,0$	$\mu(T, \varepsilon) = 0,5$
$\mu(U, A) = 0,1$	$\mu(U, C) = 0,9$	$\mu(U, G) = 0,5$	$\mu(U, T) = 0,1$	$\mu(U, \varepsilon) = 0,0$
$\mu(\varepsilon, A) = 0,5$	$\mu(\varepsilon, C) = 0,5$	$\mu(\varepsilon, G) = 0,5$	$\mu(\varepsilon, T) = 0,5$	$\mu(\varepsilon, \varepsilon) = 0,0$

Tabela 2: Valores de pertinência para as transições do Autômato Finito Nebuloso $\tilde{M}_2$.

Baseados na estrutura do autômato finito nebuloso $\tilde{M}_2$ e na Tabela 2, os exemplos seguintes mostram os graus de pertinência para diversas cadeias, bem como a seqüência de estados percorrida para chegar ao mínimo de operações necessárias:

μ(AACT, AACT) $= 1 \times 1 \times 1 \times 1 = 1$ $(Q_0, Q_1, Q_2, Q_3, Q_4)$
μ(AAUT, AACT) $= 1 \times 1 \times 0,9 \times 1 = 0,9$ $(Q_0, Q_1, Q_2, Q_3, Q_4)$
μ(TTCT, AACT) $= 0,7 \times 0,7 \times 1 \times 1 = 0,49$ $(Q_0, Q_1, Q_2, Q_3, Q_4)$
μ(AAAUT, AACT) $= Max((1 \times 1 \times 0,5 \times 0.9 \times 1), (1 \times 1 \times 0,3 \times 0,5 \times 1),$
$(1 \times 1 \times 0,3 \times 0,1 \times 0,5), ...) = 0,49$ $(Q_0, Q_1, Q_2, Q_2, Q_3, Q_4)$
μ(AAT, AACT) $= Max((1 \times 1 \times 0,5 \times 1), (1 \times 1 \times 0,3 \times 0,5),$
$(1 \times 1 \times 0,5 \times 0,5 \times 0,5), ...) = 0,5$ $(Q_0, Q_1, Q_2, Q_3, Q_4)$

5 Considerações Finais

Também há, em comparação ao mesmo modelo, um ganho aparente quanto à eficiência computacional, uma vez que não é necessário percorrer exaustivamente todas as possibilidades para obter o resultado, como acontece no pior caso dos autômatos finitos não-determinísticos. Entretanto, experimentos e estudos adicionais são necessários antes de ser possível fazer com certeza tal afirmativa.

Embora o poder computacional dos autômatos finitos (e dos autômatos finitos nebulosos) é relativamente restrito, é possível expandir através de modelos derivados, como autômatos de pilha estruturada (NETO, 1994) e os autômatos adaptativos (NETO; BRAVO, 2002) de maneira a obter um maior poder computacional com poucas alterações no modelo proposto.

Outra extensão futura possível desse trabalho é criar uma implementação computacional para o problema do reconhecimento aproximado de cadeias capaz de reduzir o custo computacional aproveitando as especificidades da estrutura do autômato finito nebuloso utilizado para fazer o reconhecimento aproximado de cadeias.

Dessa forma, tal implementação do modelo proposto nesse artigo poderia ser a base para a resolução mais eficiente de diversos problemas onde o reconhecimento aproximado de cadeias é necessário.

Referências

ALBERT, P. The algebra of fuzzy logic. *Fuzzy Sets and Systems*, v. 1, p. 203–230, 1978.

LEE, C. C. Fuzzy logic in control systems: Fuzzy logic controller - part i. *IEEE Transactions on Systems, Man and Cybernetics*, v. 20, n. 2, p. 404–418, March/April 1990.

LEE, C. C. Fuzzy logic in control systems: Fuzzy logic controller - part ii. *IEEE Transactions on Systems, Man and Cybernetics*, v. 20, n. 2, p. 419–435, March/April 1990.

LEE, H. S. Minimizing fuzzy finite automata. *9th IEEE International Conference in Fuzzy*, 2000.

LEWIS, H. R.; PAPADIMITRIOU, C. H. *Elementos de Teoria da Computação*. 2. ed. Porto Alegre: Bookman, 2000.

MATEESCU, A. et al. Lexical analysis with a simple finite-fuzzy-automaton model. *Journal of Computader Science*, 1995.

MENGE, K. Statistical metrics. *Procedures of National Academy of Science USA*, v. 28, p. 535–537, 1942.

NAVARRO, G. A guided tour to approximate string matching. *ACM Computing Surveys*, v. 33, n. 1, p. 31–88, March 2001.

NETO, J. J. Adaptive automata for context-sensitive languages. *SIGPLAN NOTICES*, 1994.

NETO, J. J.; BRAVO, C. Adaptative automata - a reduced complexity proposal. In: CHAMPARNAUD, J.; MAUREL, D. (Ed.). [S.l.: s.n.], 2002. v. 2608, p. 158–168.

NGUYEN, H. T.; WALKER, E. A. *A First Course in Fuzzy Logic*. 2. ed. New York: Chapman & Hall/CRC, 2000.

SHANNON, C. A mathematical theory of communication. *The Bell System Technical Journal*, 1948.

VOLLET, R. *Reconhecimento de Movimentos com Redes Neurais Artificiais*. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, 2001.

VOXMAN, W.; GOETSHCHEL, R. A note on the characterization of the max and min operators. *Information Sciences*, v. 30, p. 5–10, 1983.

ZADEH, L. A. Fuzzy sets. *Information and Control*, v. 8, p. 338–353, 1965.

ZADEH, L. A. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man and Cybernetics*, v. 3, p. 28–44, 1973.

Sistema Multiagentes Utilizando a Linguagem AgentSpeak(L) para Criar Estratégias de Armadilha e Cooperação em um Jogo Tipo PacMan

Alisson Rafael Appio (Titan Informática)
alissonr_appio@yahoo.com.br

Jomi Fred Hübner (FURB/DSC)
jomi@inf.furb.br

Resumo. Este artigo descreve um Sistema Multiagentes (SMA) atuando sobre um jogo tipo PacMan, onde os personagens fantasmas são concebidos como agentes. A arquitetura e a linguagem utilizadas são *Belief-Desires-Intentions* (BDI) e *AgentSpeak(L)*, respectivamente. Os agentes têm como objetivo criar estratégias de armadilha e cooperar na execução das armadilhas criadas, dificultando a vitória do personagem come-come, que é controlado por um usuário. A ferramenta ***Jason*** é utilizada para interpretar e executar o código *AgentSpeak(L)*.

Palavras-chave: Cooperação em sistemas multiagentes; jogos; estratégias de armadilha em jogos; arquitetura BDI; *AgentSpeak(L)*.

1 Introdução

Os jogos de computadores cada vez mais possuem um mercado atrativo, recebendo a atenção dos cientistas no desenvolvimento de técnicas computacionais sofisticadas com o uso de várias mídias, animações com gráficos 2D e 3D, vídeos, som, etc. A construção de jogos é uma das tarefas mais difíceis dentro da computação.

Um dos jogos mais populares é o PacMan. Neste jogo existem dois tipos de personagens, sendo eles: os fantasmas e o come-come. O personagem come-come é controlado por um usuário e tem como objetivo comer todas as bolinhas que estão situadas no cenário. Quando terminar de comer as bolinhas, o usuário vence o jogo. Os personagens fantasmas são controlados pelo computador e têm como objetivo evitar que o come-come vença o jogo. Para isso, os fantasmas tentam matar o personagem come-come.

Neste artigo, é descrito um jogo tipo PacMan, onde os personagens fantasmas são concebidos como agentes BDI, utilizando técnicas de cooperação em Sistema Multiagentes (SMA). O objetivo dos agentes é evitar que o usuário vença o jogo, uma das formas disto acontecer é criar estratégias de armadilha, prendendo o come-come em algum lugar do mundo para conseguir matá-lo e conseqüentemente não deixando que o jogador vença. Este artigo visa mostrar um sistema que utiliza agentes BDI programados na linguagem *AgentSpeak(L)*. O código *AgentSpeak(L)* é interpretado e executado pela ferramenta ***Jason***.

2 Sistema multiagentes

A área de SMA estuda o comportamento de um grupo de agentes (aplicando as técnicas de Inteligência Artificial (IA) clássica) que cooperam para resolver um problema que normalmente um único agente não seria capaz de resolver. Ocupa-se da construção de sistemas computacionais a partir da criação de entidades de software autônomas que interagem através de um ambiente compartilhado por outros agentes de uma sociedade e atuam sobre esses ambientes, alterando seu estado.

Definições mais detalhadas de SMA, seus problemas e aplicações podem ser encontradas nas seguintes referências: Alvares e Sichman (1997), Bordini, Vieira e Moreira (2001), Demazeau e Müller (1990), Jennings e Wooldridge (1998), Weiß (2000), Wooldridge (2002), Bordini e Vieira (2003).

Em SMA, existe a necessidade de coordenar as interações que ocorrem entre os agentes. Vários autores apresentam o tema coordenação, entre os quais cita-se (WEISS, 2000; WOOLDRIDGE, 2002; OLIVEIRA, 2001; RUSSEL; NORVIG, 2003). Os tipos de coordenação são descritos nas seções seguintes.

2.1 Coordenação

Pode-se dizer que coordenação em SMA é um processo no qual um agente raciocina sobre suas ações locais e ações de outros agentes com o objetivo de garantir que a comunidade funcione de maneira coerente (JENNINGS, 1996). É um ato de trabalhar em conjunto no sentido de atingir um acordo com objetivo(s) comum(ns) de forma harmoniosa. A necessidade de coordenação surge do fato da existência de dependências entre as ações dos agentes e da impossibilidade de resolução de um problema por um único agente, seja pela insuficiência de recursos, informações ou capacidade dos agentes (JENNINGS; WOOLDRIDGE, 1998).

Um exemplo de coordenação entre agentes acontece quando dois ou mais robôs precisam carregar uma mesa. Eles devem decidir quem vai pegar uma determinada parte da mesa através de uma negociação entre eles e também coordenar suas ações para conseguirem carregar a mesa, pois se eles não coordenarem suas ações um robô poderá ficar parado enquanto o outro tentará carregá-la, conseqüentemente derrubando a mesa. A coordenação acontece quando os robôs trabalham juntos para resolver um problema, ou seja, mover a mesa para outro lugar.

2.2 Cooperação

Cooperação é um tipo de coordenação entre os agentes que exercem ações com objetivo de atingir um bem social[1], *i.e.*, estão preocupados em atingir objetivos partilhados com outros agentes. Para que os agentes consigam uma cooperação satisfatória cada agente deve manter um modelo dos outros agentes e também desenvolver um modelo das futuras interações (WEISS, 2000).

Quando os agentes estão trabalhando cooperativamente, pode-se dizer que estão trabalhando como equipes e comportam-se de forma a incrementar a utilidade global do sistema e não sua utilidade individual, por exemplo:

a) cooperação entre agentes acontece em um ambiente de direção de um táxi, onde o agente deve evitar colisões para maximizar a medida de desempenho de todos os agentes (RUSSEL; NORVIG, 2003);

b) o planejamento de equipes em tênis de duplas. Dois agentes que jogam em uma equipe de tênis de duplas têm como objetivo comum vencer a partida, dando origem a vários sub-objetivos. Um dos sub-objetivos gerado é que eles tenham que devolver a bola que foi lançada para eles e assegurar que pelo menos um deles estará cobrindo a rede, pois essa é uma boa estratégia de jogo (RUSSEL; NORVIG, 2003).

Muitas vezes são adotadas “convenções” ou “leis sociais” para os planejamentos em SMA. Por exemplo, em jogos de tênis de dupla, a bola pode estar aproximadamente eqüidistante dos dois parceiros. Para resolver esse impasse, um dos agentes poderia gritar “é minha!” ou “é sua!”, através da comunicação estabelecida entre eles (RUSSEL; NORVIG, 2003).

[1]Bem social, no sentido onde os agentes não entrem em discussão sobre seus objetivos, quando estes objetivos são de interesse da sociedade como um todo

2.3 Agentes

Vários autores apresentam definições de agentes, e entre as definições mais aceitas destacam-se as de Alvares e Sichman (1997), Barone et al. (2004), Bordini e Vieira (2003). Segundo estes autores, um agente é uma entidade real ou virtual, que está inserida em um ambiente, podendo perceber, agir, deliberar e comunicar-se com outros agentes e possui comportamentos autônomos. Os agentes podem ser divididos em duas categorias: reativos e cognitivos. Agentes reativos possuem comportamentos simples, não possuindo nenhum modelo do mundo onde estão atuando e possuem comportamento estímulo-resposta. Agentes cognitivos possuem comportamentos complexos onde eles deliberam e negociam suas ações com os outros agentes. Na construção de agentes cognitivos em SMA é importante mencionar alguns aspectos, como: percepção; ação; comunicação; representação; motivação; deliberação; raciocínio e aprendizagem.

Existem diversas arquiteturas para os agentes. As arquiteturas mais conhecidas na literatura são: Reativa, *Subsumption*, BDI, Deliberativa e em Camadas. Uma arquitetura para o modelo cognitivo é a arquitetura BDI (as demais arquiteturas não são abordadas neste trabalho).

BDI é uma arquitetura caracterizada por três atitudes mentais que são as crenças, os desejos e as intenções. Os principais criadores da arquitetura BDI foram Georgeff e Rao (BORDINI; VIEIRA, 2003). A fundamentação filosófica para esta concepção de agentes vem do trabalho de Dennett (1987) sobre sistemas intencionais e de Bratman (1987) sobre raciocínio prático. Uma arquitetura BDI genérica é mostrada na figura 1.
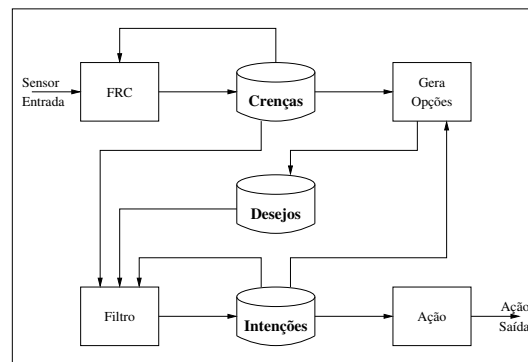
Figura 1: Arquitetura BDI genérica. Fonte: (WOOLDRIDGE, 1999)

As crenças representam tudo aquilo que o agente sabe sobre o ambiente e sobre os agentes daquele ambiente (inclusive sobre si mesmo). Os desejos representam os estados do mundo que o agente quer atingir. As intenções representam a seqüência de ações que um agente compromete-se a executar para atingir sua meta.

A função FRC (Função de Revisão de Crenças) recebe as informações do ambiente, podendo ler e atualizar a base de crenças do agente. Com as alterações do estado do ambiente, podem ser gerados novos objetivos. A função Gera Opções verifica quais estados devem ser atingidos de acordo com o estado atual e as intenções com que o agente está comprometido. A função filtro serve para atualizar o conjunto de intenções do agente com base nas crenças e desejos que ele possui. A função de Ação representa a escolha de uma determina ação para ser executada.

Uma das linguagens que considera os conceitos da arquitetura BDI é a linguagem *AgentSpeak(L)*, descrita na próxima seção.

3 Linguagem AgentSpeak(L)

Agentes cognitivos podem ser especificados através da linguagem *AgentSpeak(L)*. Um programa *AgentSpeak(L)* é especificado por um conjunto de crenças, planos, eventos ativadores e um conjunto de ações básicas que o agente executa no ambiente. Os programas feitos em *AgentSpeak(L)* são interpretados de maneira similar à programas escritos em lógica (como por exemplo, programas escritos em Prolog).

Uma crença é um predicado de primeira ordem na notação lógica usual (ou fatos, no sentido de programação lógica) e literais de crenças são átomos de crenças ou suas negações que formarão a base de crenças do agente.

Planos fazem referência a ações básicas que um agente é capaz de executar em seu ambiente, sendo composto por um evento ativador, contexto e corpo. Os planos são sensíveis ao contexto, i.e., necessitam que certas condições sejam satisfeitas para serem executados, sendo que o contexto deve ser uma conseqüência lógica da base de crenças do agente no momento em que o evento é selecionado pelo agente para o plano ser considerável aplicável. O corpo do plano é uma seqüência de ações básicas ou subobjetivos que o agente deve atingir ou testar quando uma instância do plano é selecionada para execução.

A linguagem *AgentSpeak(L)* distingue dois tipos de objetivos: objetivos de realização e objetivos de teste. Objetivos de realização e teste são predicados, tais como crenças, porém com operadores prefixados “!” e “?” respectivamente. Objetivos de realização expressam os desejos do agente e objetivos de teste retornam a unificação do predicado de teste com uma crença do agente ou pode falhar quando não existir nenhuma crença que seja satisfeita (BORDINI; VIEIRA, 2003).

Quando o agente percebe informações sobre o ambiente, é gerado um evento com esta percepção, sendo adicionado a sua base de crenças. É gerada uma lista com os planos que podem ser executados com essa percepção (i.e. que tenham o predicado do evento gerado pelo ambiente) e testado o contexto do plano para verificar quais planos podem ser aplicados. Por fim é selecionado um plano da lista de planos e o agente executa o plano selecionado.

3.1 Ferramenta *Jason*

A ferramenta ***Jason***[2] (do inglês: *A Java-based AgentSpeak Interpreter Used with Saci For Multi-Agent Distribution Over the Net*) proporciona a interpretação e execução de programas escritos na linguagem *AgentSpeak(L)*. SMA são facilmente configurados nesta ferramenta, podendo ser executado em vários computadores através do SACI[3] (HÜBNER; SICHMAN, 2000). ***Jason*** é implementado na linguagem Java (executada em múltiplas plataformas), sendo *Open Source* e distribuído sob a licença GNU LGPL.

Um SMA desenvolvido na ferramenta ***Jason***, possui um ambiente onde os agentes estão situados e um conjunto de instâncias de agentes *AgentSpeak(L)*. O ambiente dos agentes, deve ser desenvolvido na linguagem Java. ***Jason*** possui os seguintes recursos (BORDINI; HÜBNER et al., 2004):

a) negação forte (*strong negation*), portanto é possível construir sistemas que consideram mundo-fechado (*closed-world*) e mundo-aberto (*open-world*);

b) tratamento de falhas em planos;

[2] Disponível em `http://jason.sourceforge.net`
[3] Disponível em `http://www.lti.pcs.usp.br/saci/`

c) comunicação baseada em atos de fala (incluindo informações de fontes como anotações de crenças);

d) anotações em identificadores de planos, que podem ser utilizados na elaboração de funções personalizadas para seleção de planos;

e) suporte para desenvolvimento de ambientes (que normalmente não é programado em *AgentSpeak(L)*);

f) possibilidade de executar o SMA distribuidamente em uma rede (usando o SACI);

g) possibilidade de especificar (em Java) as funções de seleção de planos, as funções de confiança e toda a arquitetura do agente (percepção, revisão de crenças, comunicação e atuação);

h) possui uma biblioteca básica de "ações internas";

i) possibilitar a extensão da biblioteca de ações internas.

A configuração do SMA é feita em arquivos com extensão .mas2j, basicamente é informado qual a arquitetura do SMA, "Centralized" ou "Saci" (centralizado ou distribuída), qual o ambiente onde os agentes estão situados e os agentes. A programação dos agentes *AgentSpeak(L)* é feita em arquivos com extensão .asl. Para ver a BNF da linguagem *AgentSpeak(L)* e como configurar um SMA na ferramenta ***Jason*** pode-se consultar (BORDINI; HÜBNER et al., 2004).

4 Especificação do jogo

O *software* desenvolvido (denominado PacMan_MAS) está separado em camadas (figura 2): camada de persistência; camada de lógica do jogo e a camada de SMA (agentes).
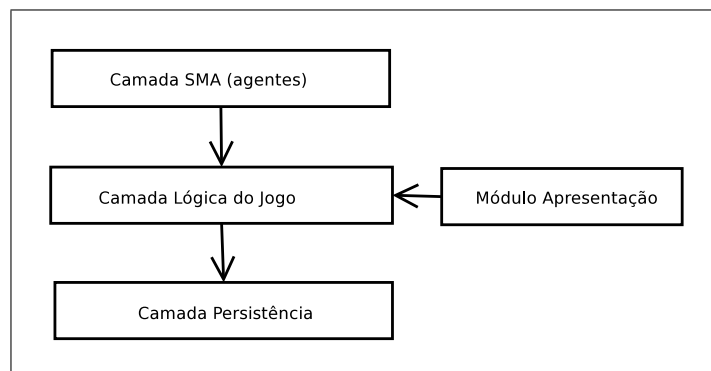
Figura 2: Camadas do jogo PacMan_MAS

A camada *Lógica do Jogo* contém a lógica de um jogo tipo PacMan, contendo as classes `GameObject` (objetos do jogo) que é uma classe abstrata, contendo os atributos que representam as coordenadas de um objeto no jogo (x e y) e possui dois métodos abstratos `int getHeight()` e `int getWidth()` que são chamados quando o jogo necessita saber a altura e largura de um determinado objeto. Ainda, é nesta classe que é implementado o método `collision(obj GameObject)` para verificar se um objeto do jogo está colidindo com outro objeto (o diagrama de classes desta camada pode ser visto na figura 3).
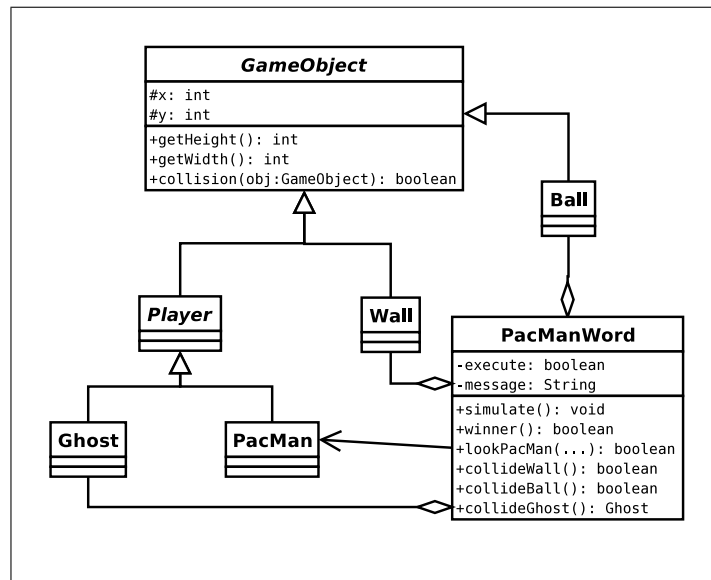
Figura 3: Diagrama de classes da camada lógica do jogo

A classe `Player` (Jogador) é uma classe abstrata, possui como super classe `GameObject`. É classe base para `PacMan` (Come-come) e `Ghost` (Fantasma) e tem como principal método, `paint(Graphics g, Component c)` que desenha a figura do jogador (come-come ou fantasma) no componente passado por parâmetro.

A classe `PacManWorld` (mundo do pacMan) representa a principal classe do jogo, possuindo listas de `Ghost` (Fantasma), `Ball` (Bola), `Wall` (Parede) e um objeto `PacMan` (Come-come). Os principais métodos dessa classe são: `simulate(); lookPacMan(String ghostName, char direction); winner(); collideWall(Player player); collideBall() e collideGhost()`.

A classe `Ball` representa uma bolinha do jogo. Possui como atributos uma imagem de uma bolinha (atributo de classe), seu estado (visível ou invisível) e os atributos herdados de `GameObject`. A classe `Wall` possui como atributos a sua largura, altura e os atributos herdados de `GameObject`.

O módulo de apresentação, desenha o jogo PacMan_MAS. Foi utilizada a estratégia de *double buffer*[4] para desenhar as bolinhas, paredes, come-come e os fantasmas.

O usuário controla a direção do come-come, pressionando as teclas *UP, DOWN, LEFT, RIGHT* que move o fantasma para cima, baixo, esquerda e direita, respectivamente.

A camada de persistência faz todo o acesso de leitura de arquivos texto contendo as coordenadas dos objetos do jogo (paredes, bolinhas, etc). O *layout* do arquivo de bolinhas, contém uma coordenada x, y. O *layout* do arquivo de paredes, possui quatro valores numéricos (coordenada x, coordenada y, a largura e a altura da parede.

Os fantasmas são movimentados através de um grafo não dirigido, cada esquina do labirinto do jogo é um vertice e as arestas são as estradas que ligam uma esquina a outra. O *layout* do arquivo de

[4]É criado uma imagem em segundo plano, feito todo o desenho do jogo nesta imagem, para depois desenhar a imagem final na tela.

vértices contém três valores numéricos (coordenada x, coordenada y e um identificador (único) de cada vértice). O *layout* do arquivo de arestas possui dois valores numéricos (identificador de origem e identificador de destino da aresta). Mais detalhes da especificação e implementação destas duas camadas podem ser obtidas em (APPIO, 2004).

5 Camada SMA

Nesta camada são implementadas as classes: `EnvironmentPacMan` para efetuar uma ponte entre o jogo (ambiente) e o interpretador ***Jason***; a classe `AgentGhost` que customiza a escolha de planos do agente; e a classe `EstadoJogo` utilizada para encontrar caminhos ótimos no labirinto do jogo.

5.1 Ambiente do SMA

Nesta camada é implementada a classe `EnvironmentPacMan` que estende a classe `Environment` (classe que a ferramenta ***Jason*** especifica) para se tornar um ambiente para o SMA, assim, efetuando uma ligação entre o ambiente e o ***Jason***.

Os principais métodos da classe `EnvironmentPacMan` são:

a) `executeAction(String ag, Term action)`, este método executa uma ação no ambiente. É passado por parâmetro o nome do agente e a ação que deve ser executada, e retorna *true* se conseguiu executar a ação ou *false* se falhar;

b) `getPercepts(String agName)`, este método é chamado por cada agente do SMA, sendo passado como parâmetro o nome do agente que está solicitando as percepções do ambiente. É retornada uma lista com as percepções (naquele instante) do agente.

Constantemente o agente solicita as suas pecepções ao ambiente e verifica se recebeu alguma mensagem de outro agente (figura 4). O agente recebe do ambiente a sua posição (coordenada x, y), através da adição da crença `pos(X, Y)`. Ele também consegue perceber o come-come (coordenada x, y), através da adição da crença `pm(X, Y)`, desde que o come-come esteja em seu campo de visão, *i.e.*, não deve existir nenhuma parede entre o come-come e o agente nas quatro direções: norte; sul; leste e oeste.
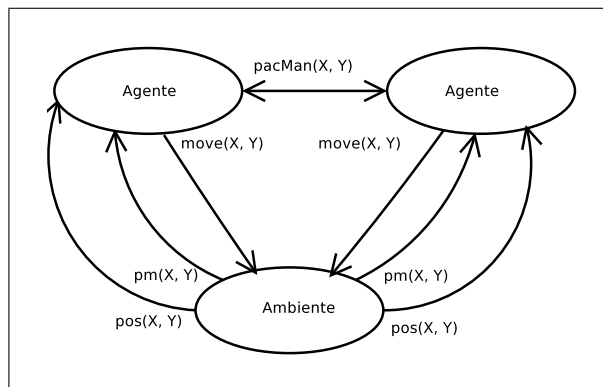
Figura 4: Comunicação entre agentes e ambiente

5.2 Especificação dos planos de movimentação do agente

Quando o agente recebe a percepção de sua posição no ambiente, por meio do evento `+pos(X, Y)`, o plano do `pos` (logo abaixo) é aplicável. Caso o agente não esteja se movendo e não esteja em modo de cooperação, o plano é selecionado e vira uma intenção do agente.

```
+ pos(X, Y) : not moving & not cooperate ←
          .nodoMaisProximo(X, Y, null, null,
             Xresult, Yresult, CornerGoal);
          + moving;
          !go(Xresult, Yresult, CornerGoal).
```

O código do plano apresentado faz o agente andar aleatoriamente. No corpo do plano, o agente executa a ação interna `.nodoMaisProximo(...)` para buscar a esquina mais próxima a ele naquele momento. É adicionada uma crença `moving` e por fim é executado o plano `go(...)` (objetivo de realização).

Para o agente descobrir uma esquina, ele executa a ação interna `.nodoMaisProximo(Xg, Yg, Xpm, Ypm, Xresult, Yresult, CornerGoal)`. Esta ação retorna as coordenadas x, y e o identificador da esquina nos parâmetros `Xresult`, `Yresult`, `CornerGoal`. Esta ação pode ser executada de duas maneiras: quando o agente está andando aleatoriamente ou quando entra em modo de cooperação (modo de cooperação é explicado a seguir). Para mover o agente de uma esquina para outra aleatoriamente (os parâmetros `Xpm` e `Ypm` não estão instanciados), é necessário efetuar uma consulta ao grafo, descobrindo quais são as esquinas vizinhas. Após descobrir quais são as esquinas vizinhas, é sorteada uma esquina aleatoriamente para simular que o agente está se movendo de uma esquina para outra ao acaso.

Para o agente mover-se de uma esquina para outra, ele executa o plano `go(X, Y, CornerGoal)` recursivamente. O primeiro plano `go(X, Y, CornerGoal)` é executado quando o agente conseguiu chegar em uma esquina passada como parâmetro, quando o contexto deste plano falha é executado o segundo plano, que possui contexto igual a *true*. O código *AgentSpeak(L)* abaixo, implementa estes conceitos.

```
+!  go(X, Y, CornerGoal):  pos(X, Y) ←
          - moving;
          + pos(X, Y).
+!  go(X, Y, CornerGoal):  true ←
          ?  pos(Xpos, Ypos);
          .nextPosition(Xpos, Ypos, CornerGoal, Xresult, Yresult);
          move(Xresult, Yresult);
          !go(X, Y, CornerGoal).
```

A ação interna `.nextPosition(...)` retorna a próxima coordenada x, y (nos parâmetros `Xresult, Yresult`) em direção a uma esquina que ele pretende ir. A esquina é representada pelo parâmetro `CornerGoal`, os parâmetros `Xpos, Ypos` representam a ultima posição do agente no ambiente.

O agente modifica o ambiente (alterar o estado do jogo) através da ação representada pelo literal `move(X, Y)`, que atualiza as coordenadas x, y do agente que executou esta ação.

5.3 Estratégia de armadilha no jogo

Quando um fantasma percebe o come-come, ele gera dois objetivos: mover-se para as coordenadas x e y do come-come (para matar o come-come) e o outro objetivo gerado é de enviar uma mensagem para os outros agentes do SMA, informando a posição do come-come. Quando um fantasma recebe uma mensagem de um outro fantasma sobre a posição do come-come, ele entra em modo de cooperação.

Ao entrar em modo de cooperação, a ação interna `.nodoMaisProximo(...)` efetua uma busca heurística, utilizando o algoritmo A* (RUSSEL; NORVIG, 2003) para saber qual é a esquina mais próxima a ele e ao come-come. É feita uma busca com heurística da menor distância em linha reta entre dois pontos. Os estados sucessores (algoritmo de busca) possuem como $custo$ a distância de uma esquina até outra. O g é o custo acumulado de todo o caminho (soma do $custo$). O h é a distância do vértice corrente até o vértice meta. O f representa a soma do $g + h$ (mais detalhes sobre o algoritmo de A* consulte (RUSSEL; NORVIG, 2003)).

O plano abaixo é selecionado quando um agente está vendo o come-come, este plano torna-se uma intenção quando é percebido o evento `+pm(X, Y)`. O agente deve comunicar-se com os outros agentes para avisar onde o come-come esta (a posição x, y do come-come) e mover-se em direção ao come-come. Para o agente se mover em direção ao come-come, ele executa a ação interna `.nodoMaisLonge(...)`, que busca a esquina mais longe em linha reta em relação ao come-come e o agente, *i.e*, fazendo o agente matar o come-come. Para o agente comunicar-se com os outros agentes ele executa o plano `sendCoordinatePacMan(X, Y, ghost, 1)`, os parâmetros x, y representam a posição do come-come, o terceiro parâmetro (`ghost`) será concatenado com um número que estiver no quarto parâmetro (o número começa com 1 e termina com 5), representando os nomes dos agentes (ghost1, ghost2, ..., ghost5).

```
+ pm(X, Y) : //...
        ← //...
        ?  pos(Xg, Yg);
        .nodoMaisLonge(Xg, Yg, X, Y, Xresult, Yresult, GoalEsquina);
        !  sendCoordinatePacMan(X, Y, ghost, 1);
        + moving;
        !  go(Xresult, Yresult, GoalEsquina).
```

A comunicação entre os agentes é feita através do envio de uma mensagem. Para isso, é usada a ação interna `.send(ghost, tell, literal)`, o primeiro parâmetro representa quem vai receber a mensagem, o segundo parâmetro representa que a mensagem é afirmativa (LABROU; FININ, 1997), o último parâmetro representa um literal que será enviado. Ao perceber o come-come, o agente tem como objetivo avisar os outros agentes da posição do come-come. Para isso, ele executa o plano `sendCoordinatePacMan(...)` (apresentado abaixo). No corpo deste plano é executada a ação interna `.send(...)` que envia a mensagem com o literal `pacMan(X,Y)`, para um determinado agente. Para evitar que o agente mande uma mensagem para ele mesmo, ele executa a ação interna `.myName(MyName)`, no contexto do plano `sendCoordinatePacMan`, esta ação retorna o nome do agente (que executou esta ação) no parâmetro `MyName`.

```
// notice that the pacMan is in position X an Y
+!  sendCoordinatePacMan(X, Y, Name, Number):  .myName(N) &
   numberAg(Num) & .concat(Name, Number, NewName) &
   N != NewName & Number < Num ←
```

```
        .send(NewName, tell, pacMan(X,Y));
        !  sendCoordinatePacMan(X, Y, Name, Number+1).
+!  sendCoordinatePacMan(X, Y, Name, Number):
   numberAg(N) & Number < N ←
        !  sendCoordinatePacMan(X, Y, Name, Number+1).
+!  sendCoordinatePacMan(X, Y, Name, Number):  true ← true.
```

Ao receber a mensagem contendo o literal `pacMan(X,Y)`, é ativado o plano abaixo, deixando o agente em modo de cooperação. O agente sabe que um outro agente está vendo o come-come na coordenada x, y, então ele executa uma busca para mover-se na direção do come-come, criando uma estratégia de armadilha.

```
+ pacMan(X,Y) : true
          ← + cooperate.
```

Os planos podem ser executados em paralelo. Para a estratégia de armadilha dar certo o agente deve preferir pelos planos de movimentação (`go(...)`) e pelos planos de percepção do come-come (`pm(...)`), para fazer isto, é necessário implementar a classe `AgentGhost` que customiza a seleção de planos do agente. Caso não seja implementado uma seleção de planos, a busca heurística pode ser executada com uma visão parcial do mundo.

6 Dificuldades encontradas

No desenvolvimento deste trabalho, uma das dificuldades encontradas foi que, por se tratar de um SMA atuando sobre um jogo (tempo real) o "estado mental" de cada agente e o ambiente estão mudando constantemente, dificultando a depuração dos agentes. A ferramenta ***Jason*** permite acompanhar as mudanças nas crenças, desejos e intenções de cada agente gerando um *log* do "estado mental de cada agente".

Outra dificuldade, está relacionada com a mudança de paradigma de programação Orientação a Objetos (OO) para orientada a agentes.

Foram encontradas poucas bibliografias sobre a linguagem *AgentSpeak(L)* (linguagem especificada a poucos anos po Rao (1996)), existindo poucas aplicações com esta linguagem. Contudo, os poucos códigos que estão disponíveis na ferramenta ***Jason*** foram de grande utilidade para compreensão da linguagem e construção deste trabalho. Alguns planos podem ser executados em paralelo, causando inconsistências nas técnicas de cooperação utilizadas caso não seja implementada uma seleção de intenção.

7 Resultados e conclusões

O principal resultado é que a estratégia de armadilha proposta neste trabalho é criada pelos agentes. Porém, quando os fantasmas estão muito longe do come-come a estrátegia não tem muito sucesso, visto que, a estratégia adotada é de se mover para uma esquina em relação ao come-come para prendê-lo em algum lugar do mundo, fica difícil para eles conseguirem prender (matar) o come-come. Outro fator que pode causar a falha na estratégia de armadilha é quando o come-come sai fora do campo de visão de um dos fantasmas e assim os outros fantasmas sairão do modo de cooperação (ver seção 5.3).

Uma limitação do *software* desenvolvido é que, não foi implementada uma negociação entre os agentes, evitando que eles tenham objetivos de ir para mesma esquina, isso pode acontecer quando a

ação interna de `nodoMaisProximo` retornar a mesma esquina (naquele momento) para os agentes. Uma forma de negociação para resolver este problema poderia ser o agente que estiver mais próximo da esquina se mover em relação a esta esquina e o outro agente deveria escolher uma outra esquina. Quando a distância para chegar na esquina for a mesma para os agentes, poderia ser feita uma escolha aleatória de duas esquinas, uma para cada agente.

Este trabalho representa uma das maiores implementações (utilizando grafo, busca heurística, cooperação, representação gráfica dos agentes *AgentSpeak(L)*, etc) que se conhece, usando a ferramenta ***Jason***. A ferramenta demonstrou-se adequada na execução de códigos *AgentSpeak(L)*. A comunicação entre os agentes é feita de maneira transparente e de alto nível para o programador, pois o ***Jason*** garante o envio e o recebimento das mensagens através da ferramenta SACI (HÜBNER; SICHMAN, 2000). O SACI permite trocar mensagens entre agentes, que podem estar distribuídos em uma rede, através do padrão *Knowledge Query Markup Language* (KQML).

O objetivo deste trabalho não era testar a ferramenta ***Jason***, mas como ela está em desenvolvimento, foram feitos vários testes (com o desenvolvimento deste trabalho), conseqüêntemente encontrando alguns *bugs* (um dos *bugs* encontrados, o código do ambiente não podia estar dentro de pacotes) que foram comunicados aos desenvolvedores da ferramenta e corrigidos.

O jogo ficou um pouco lento, em conseqüência dos fantasmas se moverem de dois em dois *pixels*. Contudo, como o *software* está separado em camadas, fazer alterações para os fantasmas se moverem mais rápido, envolve alterar a ação interna `nextPosition` e o cógido *AgentSpeak(L)*. Para desenvolver novas técnicas de cooperação para o jogo, basta apenas alterar o código *AgentSpeak(L)* e ações internas dos agentes, alterando apenas a camada SMA (agentes).

Fazer os agentes cooperarem não é uma tarefa trivial, pois os agentes devem fazer um planejamento, este planejamento pode ser centralizado ou distribuído, no caso deste trabalho é feito um planejamento distribuído.

A proposta deste trabalho, usar a arquitetura BDI e cooperação para criar armadilha no jogo tipo PacMan, mostrou-se, de forma geral bastante adequada. A arquitetura BDI proporciona construir jogos onde os personagens devem ter um certo nível de inteligência, as atitudes dos personagens estão mudando constantemente de acordo com o estado corrente do jogo e de sua “mente”.

Uma das vantagens da abordagem adotada, é que o comportamento de cada agente é muito simples, não existindo nenhum plano explícito para criar armadilhas, os planos são apenas de movimentação de uma esquina para outra. A partir da interação dos agentes surge (emerge) o comportamento de criar uma armadilha (seção 5.3).

A linguagem *AgentSpeak(L)* é muito poderosa para desenvolver aplicações onde o ambiente dos agentes muda constantemente. Os planos são facilmente construídos e ampliados. Por ser uma linguagem declarativa, a programação em *AgentSpeak(L)* se torna mais elegante, proporcionando um alto nível de abstração na especificação (crenças, desejos e intenções) do agente.

Mesmo os jogos mais sofisticados (que usam busca, possuindo a localização global da posição do come-come) de PacMan, não garantem que o computador vença o jogo, em muitos casos, os fantasmas ficam correndo atrás do personagem come-come até o fim do jogo. Neste trabalho, quando os agentes estão perto das esquinas próximas ao come-come e algum fantasma está vendo o come-come, é quase impossível o come-come fugir da estratégia de armadilha, pois a armadilha vai fechar todas as passagens pelas esquinas próxima do come-come.

Referências

ALVARES, Luiz Otavio; SICHMAN, Jaime Simão. Introdução aos sistemas multiagentes. In: MEDEIROS, Cláudia Maria Bauzer (Ed.). *Jornada de Atualização em Informática (JAI’97)*. Brasília: UnB, 1997. cap. 1, p.

1–38.

APPIO, Alisson Rafael. *Sistema Multiagentes Utilizando a Linguagem AgentSpeak(L) para Criar Estratégias de Armadilha e Cooperação em um Jogo Tipo PacMan*. Blumenau: FURB, 2004. pag. 11–49 p. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação).

BARONE, Dante Augusto Couto et al. *Sociedades artificiais*: a nova fronteira da inteligência nas máquinas. Porto Alegre: Bookman, 2004.

BORDINI, Rafael H.; HÜBNER, Jomi F. et al. ***Jason**: a java-based agentspeak interpreter used with SACI for multi-agent distribution over the net*. Manual, first release. [S.l.], Jan 2004. Disponível em: <http://jason.sourceforge.net/>. Acesso em: 20 set. 2004.

BORDINI, Rafael H.; VIEIRA, Renata. Linguagens de programação orientadas a agentes: uma introdução baseada em AgentSpeak(L). *Revista de Informática Teórica e Aplicada*, v. 10, p. 7–38, Agosto 2003. Instituto de Informática da UFRGS, Brazil.

BORDINI, Rafael Heitor; VIEIRA, Renata; MOREIRA, Álvaro Freitas. Fundamentos de sistemas multiagentes. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO (SBC2001), XX JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA (JAI), 21. *Anais...* Fortaleza-CE, Brasil: Sociedade Brasileira de Computação, 2001. p. 3–41.

BRATMAN, Michael E. *Intentions, plans and practical reason*. Cambridge, MA: Harvard University Press, 1987.

DEMAZEAU, Yves; MÜLLER, Jean Pierre. *Decentralized artificial inteligence 1*. North-Holland: Elsevier Science Publishers, 1990.

DENNETT, Daniel C. *The intentional stance*. Cambridge, MA: The MIT Press, 1987.

HÜBNER, Jomi Fred; SICHMAN, Jaime Simão. SACI: Uma ferramenta para implementação e monitoração da comunicação entre agentes. International Joint Conference, 7th Ibero-American Conference on AI, 15th Brazilian Symposium on AI (IBERAMIA/SBIA 2000), 7/15, Atibaia, São Paulo, Brazil, 2000. *Proceedings (Open Discussion Track)*. São Carlos: ICMC/USP, 2000. p. 47–56.

JENNINGS, N. R. Coordination techniques for distributed artificial intelligence. In: O'Hare G.M.P Jennings N.R. (Ed.). *Foundations of distributed Artificial Intelligence*. [S.l.]: John Wiley & Sons, Inc, 1996.

JENNINGS, Nicholas R.; WOOLDRIDGE, Michael J. (Ed.). *Agent technology*: foundations, applications, and markets. Berlin: Springer-Verlag, 1998.

LABROU, Yannis; FININ, Tim. *A proposal for a new KQML specification*. Baltimore, 1997.

OLIVEIRA, Eugênio. Agents advanced features for negotiation and coordination. In: LUCK, Michael et al. (Ed.). *Multi-agent systems and applicatons*. Prague: Springer, 2001. p. 173–186.

RAO, Anand S. AgentSpeak(L): BDI agents speak out in a logical computable language. In: WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD (MAAMAW'96), 7., 1996, Eindhoven, The Netherlands. *Proceedings...* London: Springer-Verlag, 1996. (Lecture Notes in Artificial Intelligence), p. 42–55.

RUSSEL, Stuard; NORVIG, Peter. *Artificial intelligence*: a modern approach. 2^{o}. ed. New Jersey: Prentice Hall, 2003. ISBN 0-13-790395-2.

WEISS, Gerhard. *Multiagent systems*: a modern approach to distributed artificial intelligence. Cambridge, MA: MIT Press, 2000.

WOOLDRIDGE, Michael. Intelligent agents. In: WEISS, Gerhard (Ed.). *Multiagent Systems–A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA: MIT Press, 1999. cap. 1, p. 27–77.

WOOLDRIDGE, Michael. *An introduction to multiagent systems*. New York: John Wiley & Sons, 2002.

Técnicas para Comparação e Visualização de Similaridades entre Seqüências Genéticas

Felipe Fernandes Albrecht (FURB/DSC)
albrecht@inf.furb.br

Resumo. Este artigo apresentará técnicas para comparação e visualização de similaridades de seqüências genéticas. Será exibido as características das principais técnicas, analisado seus funcionamentos, comparado o custo de execução e de espaço e a sua área de aplicação. Por fim, será apresentado um *software* que implementa as técnicas de programação dinâmica para visualização e análise de similaridades entre seqüências genéticas. Para tornar possível uma melhor visualização das similaridades foi desenvolvido um novo algoritmo para matriz de pontos com filtro.

Palavras-chave: Bioinformática, Seqüências Genéticas, Programação dinâmica, Alinhamento de seqüências genéticas, Matriz de pontos, Comparação de seqüências genéticas.

1 Introdução

A comparação e análise de seqüências genéticas é parte de um campo científico chamado bioinformática. Este campo científico é baseado numa interação entre a estatística, biologia molecular e a ciência da computação. Por causa de grandes projetos de seqüênciamento genômicos, por exemplo, o Projeto do Genoma Humano, existe um crescimento exponencial na quantidade de dados disponíveis sobre seqüências genéticas e proteínas. Os métodos tradicionais de laboratório para o estudo da estrutura e função destas moléculas não são capazes de acompanhar a taxa de crescimento de novas informações. Como conseqüência, biólogos moleculares passaram a utilizar métodos estatísticos e computacionais capazes de analisar estas grandes quantidade de dados de forma mais automatizada.

Segundo Mount (2004), alinhamento de seqüências é útil para a descoberta de informações funcionais, estruturais e evolucionárias nas seqüências biológicas. Partindo da premissa de que duas seqüências similares possuem estruturas e comportamentos similares, pode-se inferir informações sobre uma nova seqüência comparando-a a outras já previamente conhecidas.

Existem diversas formas de comparações e alinhamento de seqüências genéticas. Neste artigo, pretende-se dar uma introdução sobre os principais e mais utilizados métodos de comparação de seqüências genéticas. Iniciando do mais básico, o algoritmo de força bruta, utilização de matriz de pontos, programação dinâmica, heurísticas e modelos probabilísticos. Por fim é descrito na seção 3 um software para visualização de similaridades e alinhamentos entre seqüências e um novo algoritmo na seção 3.1 para a realização de tal visualização.

2 Algoritmos e Métodos para Comparação de Seqüências

Esta seção apresentará os principais métodos e algoritmos para comparação de seqüências genéticas.

2.1 Força Bruta

A forca bruta é provavelmente o método mais simples para a comparação de seqüências. Ele é apenas utilizado para determinar se duas seqüências são idênticas ou não, sem reportar qualquer grau de similaridade. O algoritmo é bastante simples: primeiro compara-se os primeiros elementos

```
BRUTE-FORCE-COMPARE(M, N)
1  if length[M] ≠ length[N]
2    then return DIFFERENT
3
4  for i ← 0 to length[M]
5  do if M_i ≠ N_i
6        then return DIFFERENT
7  return EQUAL
```

Quadro 1: Algoritmo força bruta para comparação de seqüências

de ambas seqüenciais, depois os segundos, os terceiros e assim sucessivamente, até encontrar um erro e retornar que as seqüências não são idênticas ou chegar no fim das seqüências e retornar que ambas são idênticas. O algoritmo está especificado no quadro 1.

Um aspecto importante a ser analisado é a quantidade de operações máximas $(O(n))$ que são necessárias para obtenção do resultado neste algoritmo. Se a seqüência é uma lista de elementos aleatórios de um alfabeto de K letras, a probabilidade de acerto, isto é, a probabilidade dos dois elementos das listas que estão sendo comparados serem iguais, em qualquer posição é $1/K$. Em seqüências com alfabetos grandes e com conteúdos aleatórios, a probabilidade de acerto será muito baixa, e normalmente serão necessárias poucas comparações para determinar que as duas seqüências são diferentes. Porem, se as seqüências não são equiprobatoriamente distribuída e o alfabeto possuir um número reduzido de elementos, como no caso do DNA, onde seu alfabeto é constituído de somente quatro letras e os padrões de repetições são comuns, o número de comparações irá aumentar significativamente, desta forma, aumentando o tempo de execução do algoritmo. No pior caso, quando as seqüências são realmente idênticas, o algoritmo de força bruta irá executar N comparações de elementos (N sendo o comprimento das seqüências), ou seja a complexidade do algoritmo é $O(n)$.

O tempo de execução deste algoritmo varia consideravelmente de acordo com a natureza das seqüências a serem analisadas e os resultados são muito simples, apenas informando se as seqüências são idênticas ou não. Sendo que na comparação de seqüências genéticas, o mais importante é o grau de homogeneidade, ou seja, o quão similares são as seqüências. Mesmo desta forma, este algoritmo não deve ser descartado, pois ele é utilizado dentro de outros algoritmos mais eficientes, principalmente nos baseados em heurísticas. Variações deste algoritmo e outros métodos de comparação exata de seqüências podem ser obtidos em (C. Charras; T. Lecroq, 1996).

2.2 Matriz de pontos

Uma matriz de pontos é primariamente um método para comparação de duas seqüenciais pela observação de possíveis alinhamentos de caracteres entra as seqüências a serem analisadas. Este método também é utilizado para encontrar repetições em seqüências genéticas e para predizer regiões no RNA que são auto complementares e possuem potencial para formar uma estrutura secundária.

Numa matriz de pontos, utiliza-se uma seqüências (M) como eixo horizontal e outra seqüências (N) como eixo vertical. Cada célula da matriz de pontos, $D(i, j)$ é resultado da comparação dos resíduos i da seqüência M com o elemento j da seqüência N. Para todo par de resíduos M_i e
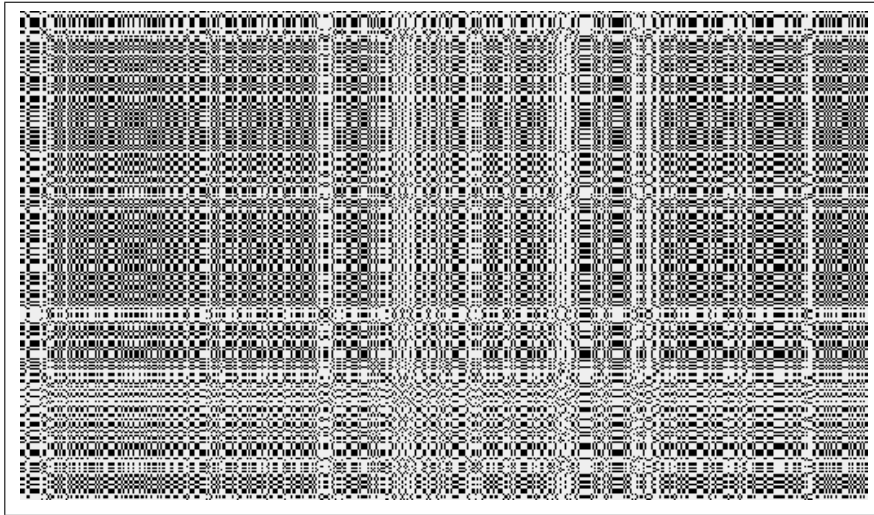
Figura 1: Comparação de duas seqüências sem utilização de filtro

```
SIMPLE-DOT-MATRIX(M, N)
1  for i ← 0 to length[M]
2  do for j ← 0 to length[N]
3      do if M_i = N_i
4            then matrix[i, j] ← 1
5            else  matrix[i, j] ← 0
6  return matrix
```

Quadro 2: Algoritmo simples para matriz de pontos

N_i idênticos, deve-se pintar a célula $D(i, j)$ correspondente de preto. Os alinhamentos locais significantes aparecem na forma de linhas diagonais bem definidas, contra um fundo de pontos sem relação. Um exemplo da sua execução pode ser observado na figura 1 e o algoritmo é especificado no quadro 2.

O tempo de execução deste algoritmo é a multiplicação do comprimento da seqüência M pela comprimento da seqüência N, resultando como tempo de execução $O(mn)$ e o espaço necessário para armazenar tal matriz de pontos é dado pela mesma multiplicação.

Segundo Mount (2004), uma das maiores vantagens do método para encontrar alinhamentos matriz de pontos, é que todos os possíveis encontros de resíduos entre as duas seqüências são exibidos, deixando ao pesquisador a escolha de identificar os mais significantes. O pesquisador irá observar as regiões melhor alinhadas ou que contenham mais repetições e poderá seleciona-las para fazer uma análise mais profunda, utilizando outros métodos de alinhamento de seqüências, como a programação dinâmica, explicada na seção 2.3.

A visualização de regiões com alinhamentos pode ser melhorada pela remoção dos encontros

```
SLIDING-WINDOW-DOT-MATRIX(M, N, WINSIZE, MIN)
 1  for i ← 0 to length[M]
 2  do for j ← 0 to length[N]
 3      do sum ← 0
 4         for w ← 0 to WINSIZE
 5         do if M_{i+w} = N_{j+w}
 6               then sum ← sum + 1
 7         if sum ≥ MIN
 8            then matrix[i][j] ← 1
 9            else  matrix[i][j] ← 0
10  return matrix
```

Quadro 3: Algoritmo para matriz de pontos utilizando janelas deslizantes

aleatórios da matriz de pontos. Esta filtragem pode ser efetuada com o uso de janelas deslizantes *(sliding window)* na comparação das duas seqüências. Ao invés de comparar uma simples posição na seqüência, toda a janela de posições adjacentes ao ponto é comparada ao mesmo tempo e o ponto é exibido apenas se um certo número de encontros exatos ocorrerem. O algoritmo é especificado no quadro 3.

Segundo Mount (2004), uma boa medida para a janela é 15 pares de bases e ser necessário 10 encontros para a exibição do ponto. De qualquer forma estes valores são arbitrários e muitas combinações podem ser utilizadas em busca de melhores visualizações.

Como pode-se observar no quadro 3, este algoritmo tem um custo de execução $O(n^3)$, tornando sua aplicação impraticável para comparações de seqüências longas. Por exemplo, considerando duas seqüências genéticas, cada qual com um comprimento de 1000 pares de bases e utilizando uma janela de 15 pares, serão efetuadas ao todo: $1000 * 1000 * 15$, resultando 150000000 comparações para a criação de uma matriz de pontos utilizando janelas deslizantes. Um algoritmo com menor custo de execução é proposto na seção 3.1.

2.3 Programação dinâmica

Quando um novo gene é descoberto, biólogos normalmente não tem idéia sobre a sua função. Uma técnica é buscar similaridades com genes com funções conhecidas e inferir a função do novo gene com base nas similaridades encontradas.

Segundo Neil C. Jones e Pavel A. Pevzner (2004), Programação Dinâmica provê um *framework* para compreender algoritmos de comparação de seqüências de DNA. A Programação Dinâmica resolve os problemas utilizando soluções previamente computadas, ou seja, dividindo o problema e reaproveitado as soluções previamente calculadas. Ela é utilizada quando as soluções dos problemas não são independentes, isto é, quando sub-problemas compartilham sub-problemas. Segundo Thomas H. Cormen, Charles E. Leiserson e Ronald L. Rivest (1999), um algoritmo de programação dinâmica soluciona todo sub-problema apenas uma vez e então salva sua resposta numa tabela, através disto evita o trabalho da recomputação da resposta toda vez que o sub-problema é encontrado.

Sendo a programação Dinâmica utilizada normalmente para otimização de problemas, no caso de alinhamentos de seqüências, a sua otimização é o alinhamento ótimo entre duas seqüências,

conforme forem especificados os parâmetros para tal. Mount (2004) complementa, o método é muito importante para analise de seqüências porque ele provê o melhor ou ótimo alinhamento entre duas seqüências.

Os algoritmos de programação dinâmica atribuem valores as mutações, deleções e substituições nas seqüências genéticas e então computa um alinhamento entre as duas seqüências que corresponde ao custo do conjunto de tais mutações. Tal alinhamento pode ser tratado como uma minimização da distância evolucionária ou a maximização da similaridades entre as duas seqüências a serem comparadas. Em ambos os casos, o custo deste alinhamento é uma medida de similaridades.

O alinhamento de duas seqüências genéticas pode ter várias soluções ótimas. O valor ótimo depende dos valores de pontuação para a casamento entre dois elementos, para o não casamento e para o *gap* (lacuna). Para o alinhamento de seqüências, há dois algoritmos principais que utilizam programação dinâmica. Para alinhamento global, onde deseja-se alinhas todos os elementos da seqüências M com todos da seqüência N, utiliza-se o algoritmo de Needleman-Wunsch e para alinhamento local, onde dá-se uma ênfase em alinhamentos em uma região, utiliza-se o algoritmo de Smith-Waterman.

2.3.1 Alinhamento global (algoritmo de Needleman-Wunsch)

O alinhamento global de duas seqüências é obtido através de construção de uma matriz de valores, onde cada célula representa um encontro entre uma base ou um gap da seqüência M e uma base ou um *gap* da seqüência N. As pontuações nas células podem ser negativas, positivas ou zero. Os valores nas última célula das linhas e colunas da matriz representam as pontuação de possíveis alinhamento. A recorrências matemática é visto na equação 1 e um possível algoritmo é apresentado no quadro 4. A função δ na equação 1 representa o valor do encontro entre os dois símbolos, retornando o valor de um encontro ou desencontro, conforme o ocorrido.

$$S_{i,j} = max \begin{cases} S_{i-1,j} - GAP, \\ S_{i,j-1} - GAP, \\ S_{i,j} + \delta(m_i, n_j). \end{cases} \tag{1}$$

2.3.2 Alinhamento local (algoritmo de Smith-Waterman)

O alinhamento local é muito similar ao alinhamento global, a diferença é que ele valoriza alinhamentos internos entre as duas seqüências e não o alinhamento completo entre elas. Esta diferença pode ser vista na equação 2, onde percebe-se a inclusão de uma nova opção de valor para as células, o zero. Então nas células da matriz de alinhamento local, os valores variarão de 0 a ∞.

$$S_{i,j} = max \begin{cases} S_{i-1,j} - GAP, \\ S_{i,j-1} - GAP, \\ S_{i,j} + \delta(m_i, n_j), \\ 0. \end{cases} \tag{2}$$

```
GLOBAL-ALIGNMENT(M, N)
 1  for i ← 0 to length[M]
 2  do matrix[i, 0] ← i * GAP_VALUE
 3  for j ← 0 to length[N]
 4  do matrix[0, j] ← j * GAP_VALUE
 5  for i ← 1 to length[M]
 6  do for i ← 1 to length[N]
 7      do if m_i = n_j
 8            then matrix[i][j] ← matrix[i − 1][j − 1] + MATCH_VALUE
 9            else  matrix[i][j] ← matrix[i − 1][j − 1] + DISMATH_VALUE
10         if matrix[i][j] < matrix[i, j − 1] + GAP_VALUE
11            then matrix[i][j] ← matrix[i][j − 1] + GAP_VALUE
12         if matrix[i][j] < matrix[i − 1, j] + GAP_VALUE
13            then matrix[i][j] ← matrix[i − 1][j] + GAP_VALUE
14  return matrix
```

Quadro 4: Algoritmo de alinhamento global utilizando programação dinâmica

2.3.3 Traceback

Para visualizar os alinhamentos, tanto globais como locais, a principal técnica utilizada é o *Traceback*. Onde cada célula possui um ponteiro para a célula que foi utilizado para calcular o seu valor.

Após todo o cálculo do alinhamento, deve-se percorrer este grafo dirigido acíclico e em cada passagem por uma célula, emitir os símbolos que correspondem a ela. O custo de execução deste algoritmo é $O(n + m)$.

Outra possível técnica, é ir na célula de valor ótimo do alinhamento e computar quais foram as células utilizadas para calcular os valores do alinhamento. Estando na célula $V(i, j)$, percorre-se as células vizinhas, $V(i-1, j-1)$, $V(i-1, j)$ e $V(i, j-1)$ e computa-se qual foi utilizada para chegar ao seu valor e então emitir o seu símbolo correspondente e executar a mesma operação nela. Desta forma, deve-se iniciar na ultima célula do alinhamento e percorrer a matriz de valores até chegar na primeira linha ou coluna. A vantagem desta técnica é que os dados requeridos para tal alinhamento já foram computados previamente e não há mais gasto com espaço.

Os algoritmos que utilizam programação dinâmica são muito exatos e retornam os melhores alinhamentos dependendo apenas dos parâmetros. A grande dificuldade deles é o espaço e tempo requerido para a sua computação, $O(mn)$. Por exemplo, para comparação de duas seqüências com dois mil pares de bases e utilizando 1 *byte* para cada par de base, serão necessários aproximadamente 30 *megabytes* para a construção da matriz. Técnicas como Dividir e Conquistar (Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest, 1999; Neil C. Jones; Pavel A. Pevzner, 2004) diminuem o espaço requeridos para $O(min(m, n))$. Para a implementação, utilizar técnicas de compactação, por exemplo, armazenar quatro nucleotídeos em um *byte*, desta forma reduz o espaço requerido em até 16 vezes.

2.4 Heurísticas

Conhecendo um problema, pode-se utilizar algumas aproximações que facilitarão a resolução deste problema. Utiliza-se informação e intuições a respeito da instância do problema e da sua estrutura para resolvê-lo de melhor forma. Sendo uma aproximação, nem toda heurística tem uma razão de qualidade comprovada matematicamente ou uma prova formal de convergência.

Para comparação de seqüências as duas principais famílias de algoritmos que utilizam heurística são FAST(Lipman, D.J.; Pearson, W.R, 1985), e BLAST (Basic Local Alignment Search Tool)(Altschul, S. F. et al., 1990).

2.5 Algoritmos probabilísticos

O principal algoritmo probabilístico para o alinhamento e busca de semelhanças entre seqüências é o Modelos Ocultos de Markov (Hidden Markov Models, HMM). Koski (2001) diz que os Modelos Ocultos de Markov foram primeiramente aplicados no reconhecimento de fala e atualmente são amplamente usados na biologia computacional e na bioinformática. Koski complementa que uma importante característica da modelagem dos modelos ocultos de Markov é a flexibilidade e adaptabilidade.

Os Modelos Ocultos de Markov tendem a funcionar num tempo de execução e de espaço menor que a programação dinâmica, porem, a qualidade dos resultados depende fortemente dos dados que foram utilizados para treinar os modelos. Deste modo ganha-se em custo do tempo de execução e espaço, mas a certeza da qualidade dos resultado é inferior.

3 Software para visualização e comparação de seqüências

Esta seção apresenta um *software* que tem o objetivo de dar ao usuário a opção de visualizar similaridades entre seqüências genéticas através de uma matriz de pontos e poder selecionar as sub-seqüências que o convém para realizar alinhamentos entre elas.

Um dos principais requisitos não funcionais, é a que ele seja simples de operar, isto é, sem menus e muitas opções de configurações. O usuário deve apenas abrir os arquivos com as seqüências genéticas e selecionar a seqüência que lhe convém. Outro requisito, é a necessidade de poder comparar seqüências com até dois mil pares de bases cada uma num tempo hábil.

Para o desenvolvimento foi o utilizado a Linguagem Java Versão 1.5, junto com a biblioteca BioJava(BIOJAVA, 2005). Esta biblioteca foi utilizada para facilitar a leitura dos dados no formato FASTA, comparação de nucleotídeos e exibição de dados.

3.1 Algoritmo para matriz de pontos

Uma dificuldade encontrada na exibição das matrizes de pontos com filtros descrita na seção 2.2, foi o seu alto custo computacional. Conforme visto na seção 2.2, o algoritmo de Matriz de Pontos com Filtros de Janelas Deslizantes tem o custo de execução $O(n^3)$. Com tal custo de execução, a execução deste algoritmo é impraticável com seqüências longas. Para resolver este problema foi criado um novo algoritmo, chamado de "Matriz de Pontos com Pontuação Mínima".

O algoritmo de Matriz de Pontos com Pontuação Mínima é baseado na programação dinâmica. Nele cria-se uma matriz onde são armazenados os valores do casamento ou não casamento entre os símbolos das posição M_i e M_j correntes. É utilizado no algoritmo um limite superior e um inferior para a pontuação das células. Estes limites são utilizados para que um trecho de duas sub-seqüência onde ocorra vários encontros ou desencontros não interfira nas pontuações seguintes. Um terceiro limite é utilizado para a exibição do ponto, ou seja, para a exibição de tal ponto, um número

```
MINIMUM-PONTUATION-DOT-MATRIX(M, N, LIM)
 1  for i ← 0 to length[M]
 2  do if m_j = n_0
 3        then matrix[i, 0] ← 1
 4        else  matrix[i, 0] ← 0
 5  for i ← 0 to length[N]
 6  do if m_0 = n_i
 7        then matrix[0, i] ← 1
 8        else  matrix[0, i] ← 0
 9  for i ← 1 to length[M]
10  do for j ← 1 to length[M]
11     do if m_i = n_j
12           then if matrix[i − 1][j − 1] + 1 < UPPER_LIMIT
13                 then matrix[i][j] ← matrix[i − 1][j − 1] + 1
14                 else  matrix[i][j] ← matrix[i − 1][j − 1]
15           else  if matrix[i][j] > 0
16                 then matrix[i][j] ← matrix[i − 1][j − 1] − 1
17                 else  matrix[i][j] ← 0
18  return matrix
```

Quadro 5: Algoritmo para matriz de pontos utilizando programação dinâmica

mínimo de casamentos devem ter ocorrido anteriormente. A recorrência matemática desde algoritmo é exibido na equação 3 e o algoritmo especificado no quadro 5.

$$S_{i,j} = min \begin{cases} UPPER_LIMIT, \\ max \begin{cases} S_{i-1,j-1} + \delta(m_i, n_j), \\ 0. \end{cases} \end{cases} \qquad \delta(m_i, n_j) = \begin{cases} 1 \, if \; m_i = n_j, \\ -1 \, if \; m_i \neq m_j. \end{cases} \tag{3}$$

A fórmula matemática apresenta uma recorrência, onde toda a célula $V[i, j]$ necessita do valor da célula $V[i - 1, j - 1]$ para computar o seu valor. O algoritmo primeiramente inicializa os valores da linha 0 e coluna 0, sendo que estes não possuem células anteriores para o cálculo do seu valor. Após, para cada célula $V[i, j]$, o algoritmo checa se ocorreu um casamento entre os pares de bases M_i e N_j, caso não ocorra, o valor da célula $V[i, j]$ será o valor da célula $V[i - 1, j - 1] - 1$. Porém, como observado na equação 3, há um limite inferior de 0 para cada célula. Caso ocorra um casamento entre os pares de bases, o valor da célula $V[i, j]$ será o valor da célula $V[i - 1, j - 1] + 1$. Conforme a equação 3, há um limite superior, determinando que o valor de cada célula nunca poderá ser superior a UPPER_LIMIT.

Como este algoritmo percorre todos os elementos da seqüência M e compara com todos os elementos da seqüência N, seu custo de execução é $O(MN)$ e por utilizar uma matriz de tamanho M por N para armazenar os sub-resultados, seu custo de espaço é $O(MN)$. Desta forma, o custo de tempo de execução e de espaço do algoritmo possui um crescimento de ordem quadrática (n^2), inferior ao custo de execução do algoritmo de janelas deslizantes da seção 2.2, que é $O(n^3)$.
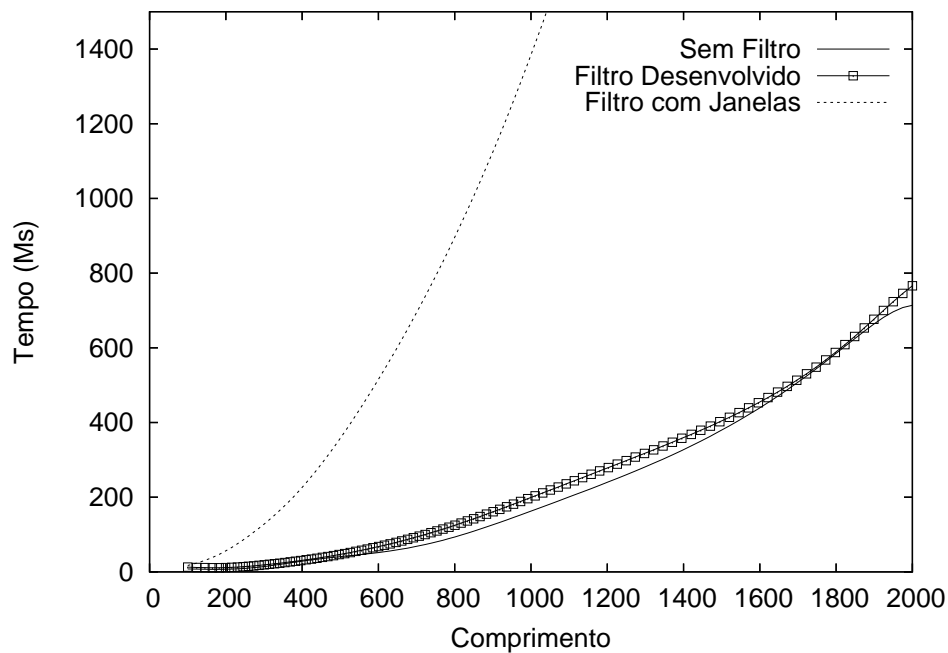
Figura 2: Comparação do tempo de execução dos Algoritmos.

Um gráfico comparando execuções dos algoritmos num computador *Atlhon XP 1700+* com 380 *megabytes* de memória e executando *Fedora Core 4* é exibido na figura 2. Nele é exibido o tempo de execução do algoritmo sem filtro, com filtro de janelas deslizantes apresentado na seção 2.2 e com o filtro desenvolvido, que utiliza o algoritmo de Matriz de Pontos com Pontuação Mínima. Observa-se que o tempo de execução do novo algoritmo aproximasse em muito do algoritmo sem filtro e conseqüentemente fica muito abaixo do tempo de execução do algoritmo com Filtro de Janelas Deslizantes. Enquanto uma matriz que contem duas seqüências com o comprimento de 1000 nucleotídeos demora aproximadamente $1,5$ segundos para serem filtradas no algoritmo de com Filtro de Janelas Deslizantes, no algoritmo de Matriz de Pontos com Pontuação Mínima a filtragem é executado em aproximadamente 200 milésimos de segundo. Para filtrar uma matriz com duas seqüências, cada uma com o comprimento de 2000 nucleotídeos, o algoritmo de janelas deslizantes executa em aproximadamente $4,6$ segundo, o novo algoritmo efetua a operação em aproximadamente 800 milésimos de segundo.

A qualidade dos resultados do Algoritmo de Matriz de Pontuação Mínima é igual ou superior ao do Algoritmo de Janelas Deslizantes. Porque enquanto as janelas deslizantes analisam apenas o conteúdo da janela atual, o Algoritmo de Matriz de Pontuação Mínima utiliza todos os dados calculados da diagonal dos pares de bases atuais.

O algoritmo retorna uma matriz de valores, que necessitam ser filtrados antes de serem exibidos. Este filtro é feito através de um limite mínimo que é utilizado no algoritmo especificado no quadro 6. Conforme descrito na seção 3.1, o algoritmo de exibição da matriz com o resultado do filtro serve para que apenas pontos que procedam um certo número de encontros sejam exibidos. Caso o valor mínimo do filtro seja 1, todos os encontros serão exibidos, como se a matriz não tivesse sido filtrada pelo algoritmo de Matriz de Pontos de Pontuação Mínima.

```
SHOW-PONTUATION-DOT-MATRIX(M, N, MIN)
1  for i ← 0 to length[M]
2  do for i ← 0 to length[N]
3      do if matrix[i][j] ≥ MIM
4            then PRINT_DOT(i, j)
```

Quadro 6: Algoritmo para exibição da matriz de pontos
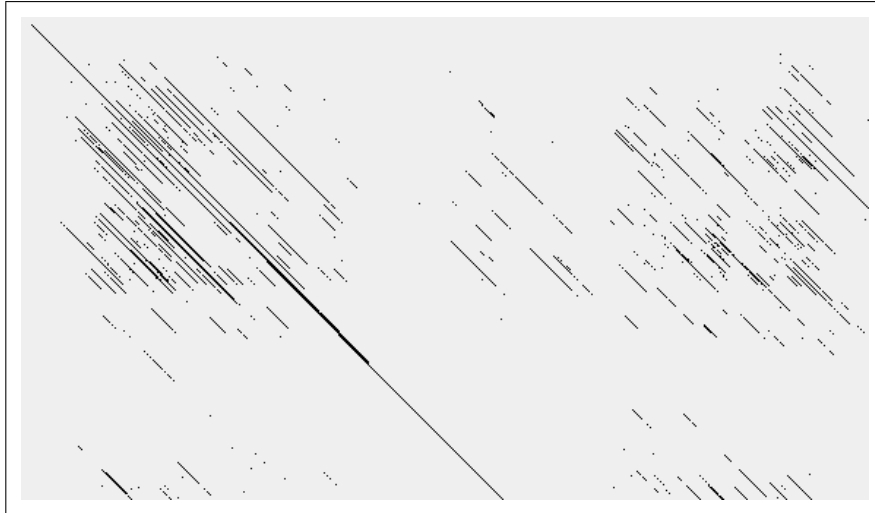
Figura 3: Comparação de duas seqüências utilizando o algoritmo de pontuação mínima

Através de testes, chegou ao valor mínimo para exibição de 7 pontos e um valor máximo de 11 pontos. Uma exibição de parte de uma comparação de seqüências genéticas utilizando o filtro de Matriz de Pontos com Pontuação Mínima com estes valores especificados é exibido na figura 3. Na imagem dois trechos de duas seqüências de animais da família das Abelhas são exibidos. Pode-se visualizar várias retas formadas na imagem. Cada reta desta imagem significa que há um alto grau de similaridade entre as seqüências naquele trecho. Por exemplo, se a reta começa na posição 40 da seqüência M (seqüência horizontal) e termina na posição 90 e começa na posição 70 da seqüência N (seqüência vertical) e termina na posição 120, signifi/ca que a sub-seqüência 40 a 90 da seqüência M e sub-seqüência 70 a 120 da seqüência$N$ são muito similares.

4 Conclusão

Muitos são os métodos e algoritmos para comparação de seqüências genéticas, compara-los e escolher um melhor é uma tarefa difícil. Cada técnica possui suas vantagens e desvantagens e principalmente sua área de aplicação. As heurísticas apresentadas na seção 2.4 são mais rápidas e requerem menos espaço para comparar e encontrar seqüências similares do que programação dinâmica, porém elas são menos sensíveis a pequenas diferenças. Estas características fazem delas úteis em

busca em banco de dados de seqüências por similaridades. Por exemplo, uma nova versão(Altschul, S. F.; Scheffer, A. A., 1997) do algoritmo BLAST, é utilizando num dos maiores banco de dados de seqüências genéticas do mundo, o NCBI(NCBI, 2005). O Modelo Oculto de Markov é útil quando se tem um bom conjunto de seqüências para treinar o modelo e deseja predizer algumas informações sobre a seqüência, por exemplo, se ela é codificante, ou qual será a família da proteína.

Para a comparação de duas seqüências que possuem alguma similaridade previamente conhecida, é interessante uma comparação mais detalhada, neste caso uma matriz de pontos para a visualização das similaridades e alinhamentos que utilizam os algoritmos vistos na seção 2.3 são úteis e funcionais para este tipo de comparação. O grande problema das comparações genéticas é o grande volume de dados, e isto torna a utilização de algoritmos de crescimento experiências difíceis de serem aplicados, tornando necessário a utilização de heurísticas ou modelos probabilísticos para a busca de similaridades.

O algoritmo e o *software* [1] apresentados proporcionam ao usuário a opção de visualizar similaridade de seqüências num tempo hábil e tendo a sua disposição o melhor detector de padrões existente, o cérebro. Conforme visto na figura 3, o algoritmo de Matriz de Pontos com Pontuação Mínima da ao usuário a opção de visualizar as partes mais semelhantes entre as seqüências e então o usuário pode seleciona-las para alinha-las utilizando o algoritmo de alinhamento global visto na seção 2.3.1.

Referências

Altschul, S. F. et al. Basic local alignment search tool. *Journal of Molecular Biology*, v. 215, p. 403–410, 1990.

Altschul, S. F.; Scheffer, A. A. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, v. 25, n. 17, p. 3389–3402, 1997.

BIOJAVA, Comunidade. *BioJava*. 2005. Disponível em: <http://www.biojava.org>. Acesso em: 1 Ago. 2005.

C. Charras; T. Lecroq. *Exact string matching algorithms*. 1996. Disponível em: <http://www-igm-.univ-mlv.fr/~lecroq/string/>. Acesso em: 20 Ago. 2005.

KOSKI, Timo. *Hidden Markov Models For Bioinformatics*. Dordrecht: Kluwer Academic Publishers, 2001.

Lipman, D.J.; Pearson, W.R. Rapid and sensitive protein similarity searchers. *Science*, v. 227, p. 1435–1441, 1985.

MOUNT, David W. *Bioinformatics*. Cold Spring Harbor: Cold Spring Harbor Laboratory, 2004.

NCBI. *National Center for Biotechnology Information*. 2005. Disponível em: <http://www.ncbi-.nlm.nih.gov>. Acesso em: 1 Ago. 2005.

Neil C. Jones; Pavel A. Pevzner. *An Introduction to Bioinformatics Algorithms*. Cambridge: The MIT Press, 2004.

Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest. *Introduction to Algorithms*. Cambridge: The MIT Press, 1999.

[1] O *download* do *software* pode ser feito em *http://www.inf.furb.br/ albrecht/interesses/bioinformatica/seminco2005/*

Desenvolvimento de Software para Controle de Potência de Pseudo-satélite Utilizando a Técnica "User Feedback"

Luiz Eduardo Guarino de Vasconcelos (CTA-IAE)
legv@ig.com.br

Durval Zandonadi Júnior (CTA-ITA)
durval@ita.br

Fernando Walter (CTA-ITA)
fw@ita.br

Resumo: Este artigo mostra as informações técnicas de suporte à interface dos ensaios realizados em um protótipo de Pseudo-Satélite (PS). O aplicativo desenvolvido para controle automático de potência do PS adquire dados, de receptores GPS e provê realimentação para um usuário através de uma interface de comunicação no protocolo RS-232. O projeto "Pseudo-Satélite" está em desenvolvimento no Instituto Tecnológico da Aeronáutica (ITA) e faz parte de uma tese de doutorado. Este projeto consiste na proposta de uma nova técnica de superação do problema "*Near/Far*" concernente ao emprego de PS, bem como construção de protótipos e validação mediante ensaios em campo. Os testes em campo do PS e o desenvolvimento do aplicativo fazem parte do projeto e foram realizados na Divisão de Ensaios em Vôo (AEV) do Instituto de Aeronáutica e Espaço (IAE) do Centro Técnico Aeroespacial (CTA) que dispõe da infra-estrutura necessária ao desenvolvimento e ao suporte dos testes.

Palavras-chave: pseudo-satélite, GPS, near/far; RS-232, controle de potência, satélite.

1 Introdução

O Pseudo-Satélite (PS) é um transmissor de sinais GPS, localizado próximo ao solo, em posição previamente conhecida. Ele transmite seus sinais a usuários participantes de um sistema de acréscimo baseado em solo conhecido como GBAS *(Ground Based Augmentation System)*.

A utilização de PS constitui-se numa fonte adicional dos sinais de navegação para os usuários. As informações enviadas pelos PS juntam-se às enviadas pelos satélites visíveis ao receptor GPS do usuário. Esta fonte aumenta a confiabilidade, a integridade e a exatidão da solução de navegação.

A melhora de exatidão das medidas de posição, de velocidade e de tempo (PVT) obtidas com o emprego de PS em sistemas aeronáuticos, permitirão o desenvolvimento de sistemas de pouso e decolagem automáticos (ATC) na categoria III, especialmente na III-C [FAA,1984].

2 Conceitos

2.1 Caracterização do problema "NEAR/FAR"

O nível de potência transmitida por um PS é um quesito crítico, sendo um dos mais importantes, senão o mais importante a ser considerado no projeto [COBB, 1997]. Há um impasse clássico conhecido como problema "*near/far*". Quando o receptor se encontra longe do PS, a potência transmitida deverá ser suficiente para assegurar a recepção regular do sinal. Por outro lado, quando o receptor estiver próximo ao PS, nenhuma saturação ou ofuscamento dos sinais dos

satélites GPS deverá ocorrer, caso estes quesitos não se cumpram, caracteriza-se o problema *"near/far"*.

Algumas estratégias de mitigação são conhecidas desde de 1994 [Klein e Parkinson, 1994] e várias têm sido as técnicas propostas e empregadas desde então para contornar esse problema. Quaisquer que sejam essas estratégias há sempre aspectos positivos e negativos quanto a sua utilização, envolvendo, por exemplo, a necessidade de se implementar grandes alterações nos receptores GPS.

2.2 Técnica "Realimentação do Usuário"

Uma nova estratégia para se contornar o problema *near/far*, denominada de "Realimentação do Usuário" (RU - *User Feedback*), foi proposta juntamente com o desenvolvimento do projeto do PS do ITA (Figura 1).

Entende-se por faixa dinâmica de um receptor a relação entre os níveis de potência de sinal recebido mais forte e mais fraco que permitam a demodulação sem distorção e ruídos excessivos. O problema *near/far* decorre precisamente da limitação desta faixa nos receptores GPS. Este problema é inerente à rádio-difusão em geral. A RU consiste em dosar dinamicamente a potência transmitida pelo PS de modo que o sinal recebido se encontre sempre dentro da faixa dinâmica do receptor. A informação do nível de potência do sinal recebido deverá ser realimentada ao PS para que seja controlada a potência transmitida, a qual depende da distância em que o usuário se encontra.
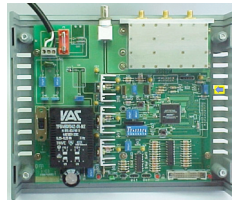
Figura 1: Foto do protótipo do Pseudo-Satélite GPS em desenvolvimento no ITA

O desenvolvimento do aplicativo de controle da potência transmitida pelo PS, foi feito de forma a assegurar que a potência recebida num receptor GPS tenda à potência de referência (P_{REF}) imposta pelo operador do PS. Este patamar poderá ser modificado dinamicamente pelo operador.

O controlador de potência precisa conhecer qual a intensidade da potência recebida pelo receptor do usuário (P_{REC}), expressa em termos da relação C/N_0 em dB·Hz, referente aos sinais dos satélites e sobretudo ao PS. Estão implícitos nesta relação efeitos de ganhos e descasamento de polarização das antenas, obstruções de sinal e perturbações no receptor. Em outras palavras, o parâmetro C/N_0 reporta efeitos e perturbações tanto do canal de transmissão como da própria recepção. A correta atuação do controlador deve ocorrer mesmo existindo perturbações no canal de transmissão e na recepção, mesmo que este canal seja variante no tempo.

Para se atender aos requisitos, foi elaborado um controlador implementado na forma de aplicativo dedicado, especialmente desenvolvido para esta finalidade, a ser executado num microcomputador PC. O PS dispõe de uma porta de comunicação serial por onde o microcomputador PC envia comandos de potência oriundos do programa controlador de potência. A Figura 2 ilustra a malha de controle proposta.
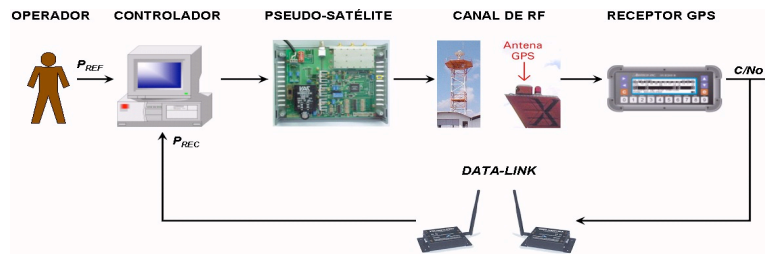
Figura 2: Malha básica de controle envolvendo pseudo-satélite e controlador de potência transmitida.

3 Desenvolvimento do software

3.1 O Controlador de potência

O aplicativo denominado UF ("*User Feedback*") foi desenvolvido pela Subdivisão de Telemetria (AEV-T) da Divisão de Ensaios em Vôo (AEV) do Centro Técnico Aeroespacial (CTA), para auxiliar na solução do problema *near/far*. Este módulo tem, basicamente, o objetivo de adquirir os dados do receptor GPS, aplicar as estratégias de controle de potência e realimentar o PS, possibilitando o controle automático da potência. Este controle é realizado através de interfaces RS-232 que realizam a comunicação com o receptor GPS e com o PS. Desenvolvido na linguagem de programação *Microsoft Visual Basic® 6.0 for Windows,* este aplicativo utiliza o componente MWStrip.ocx para visualização dos gráficos em tempo-real e o componente MSComm para controle da comunicação nas portas seriais.

3.2 Funcionamento do UF

A Figura 3 mostra o fluxograma simplificado do controlador de potência do PS. O aplicativo deve iniciar variáveis relativas à potência de referência (P_{REF}) e à potência inicialmente transmitida (P_T). Esta última é mapeada indiretamente em termos da palavra digital de controle de potência denominada de *Word_Power* (*WP*) relativa ao conversor D/A. O controlador recebe informações realimentadas do receptor GPS, dentre as quais está a potência recebida do PS (P_{REC}). Ele realiza comparação entre os valores P_{REC} e P_{REF} e em função do resultado obtido (menor, igual ou maior) calcula o novo valor ideal de potência transmitida (P_T) e o traduz em novo valor de *WP* a ser programado no conversor D/A. Este valor é transmitido através da interface RS-232 ao PS, reajustando-se assim o conversor D/A. Por fim, um arquivo que contém as informações relativas ao processo é gerado (LOG), atualizado e armazenado.

3.3 Estratégias de recuperação de potência

Estratégias foram implementadas com a finalidade de auxiliar a recaptura do sinal perdido pelo receptor GPS, ocasião esta em que não há realimentação de nível de potência recebida pelo receptor. Assim sendo, o controlador fica inoperante, e neste caso, a principal estratégia empregada, consiste na imposição de um nível de potência transmitida após a perda do sinal, cujo valor é baseado na potência média aplicada momentos antes da perda de sinal, quando o controle estava operacionalmente confiável.

3.4 Limitação de satélites

Em razão da limitação da banda do rádio-modem, o número de satélites observados pelo receptor GPS é limitado em função da taxa de aquisição dos dados.

Em função dos algoritmos internos de operação do rádio-modem utilizado, existe um pequeno atraso (entre 60 e 80 ms) de transmissão do pacote de dados. O pacote de dados é tão maior quanto mais satélites forem capturados.

Considerando-se uma taxa de aquisição fixa, há um número máximo de *bytes* que podem ser enviados ao outro extremo do canal sem que haja congestionamentos. Foram realizados ensaios com o rádio-modem a fim de se avaliar o compromisso entre o número de *bytes* a serem transmitidos e o tempo total para que o último *bit* fosse recebido, para taxas de 9600 e 38400 bps, descritos a seguir. O rádio-modem utilizado suporta até 57600 bps, porém, a escolha da taxa de 38400 bps foi adotada em conformidade com a taxa do receptor GPS.
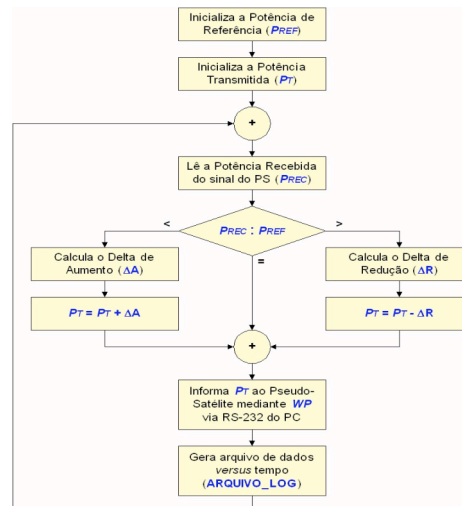
Figura 3: Fluxograma do algoritmo de recepção da potência e transmissão de nova potência ao PS.

3.5 Controle dos limites relativo e absoluto

Os limites absolutos impõem uma excursão máxima de potência permitida ao PS. Estes são compatíveis com a capacidade do próprio PS bem como com a realidade da região de utilização do PS (alcance, nível de interferência aceitável). Os limites relativos permitem a implementação de controle saturado, ou seja, pode-se limitar o degrau de atuação, impedindo-se acréscimos ou reduções bruscas de potência, o que pode levar o receptor GPS a perder o sincronismo de sinal. A implementação desses limites está relacionada com a estratégia de controle empregada, bem como com a faixa dinâmica dos receptores GPS utilizados.

Estes limites servem, portanto, de balizadores dinamicamente ajustáveis, capazes de manter os limites do controlador dentro de margens compatíveis com a realidade do ambiente e do receptor. Os limites têm efeito sempre que o controle automático estiver ativo.

3.6 LOG

O arquivo de armazenamento de dados, denominado "LOG", tem por finalidade registrar as informações mais importantes durante os ensaios com o PS. Sua análise pós-ensaio permite a avaliação do desempenho do PS, do rádio-modem, do controlador e dos satélites recebidos. O arquivo "LOG" contém os resultados dos ensaios, permitindo assim a avaliação quantitativa do desempenho deste sistema (vide Seção 8).

Taxa de Amostragem (am/s)	Tempo Decorrido do Ensaio (min)						
	1 min	5 min	10 min	20 min	30 min	45 min	60 min
1	30 kB	150 kB	300 kB	0,6 MB	0,9 MB	1,35 MB	1,8 MB
2	60 kB	300 kB	600 kB	1,2 MB	1,8 MB	2,7 MB	3,6 MB
2,5	75 kB	375 kB	750 kB	1,5 MB	2,2 MB	3,3 MB	4,5 MB
3,3	100 kB	500 kB	1 MB	2 MB	3 MB	4,5 MB	6 MB
5	150 kB	750 kB	1,5 MB	3 MB	4,5 MB	6,75 MB	9 MB
10	300 kB	1,5 MB	3 MB	6 MB	9 MB	13,5 MB	18 MB

Tabela 1: Estimativa de tamanho do arquivo LOG em kb: (500 bytes por amostra x número de amostras/segundo * tempo de coleta de dados em segundos) / 1024. Considerando-se um número de bytes adicional por conta de protocolos de gravação de arquivos, será usado o denominador 1000 ao invés de 1024 para simplificar.

3.7 Aplicação de CheckSum

Para se verificar a consistência da transmissão de dados, foi utilizado o algoritmo de *Checksum*. Neste caso o transmissor adiciona, normalmente ao final do pacote a ser transmitido, um ou mais *bits* que fazem parte do *checksum*. Este algoritmo pode ser implementado de diversas formas, desde somas primitivas até sofisticados esquemas de codificações de *bits*, capazes de indicar, em certos casos, onde o erro ocorreu. As mensagens da Associação Nacional Marítima de Navegação (NMEA - citadas na seção 5.1), retornadas pelo receptor, incorporam dois *bytes* de *checksum*, que transportam informações à cerca dos dados precedentes. Uma conferência destes *bytes* com o restante do pacote devidamente processado pode dar uma indicação, com alto grau de confiança, de que não houve erros na recepção dos *bits*. Por outro lado, a detecção de erro no pacote pode permitir seu descarte, realizando um controle mais confiável. Aplicamos este algoritmo de verificação através de somas primitivas, garantindo a qualidade e confiabilidade das informações transmitidas pelo receptor GPS.

3.8 Inserção de ruído manual

Durante a fase de desenvolvimento do aplicativo, para se simular o comportamento de um receptor móvel que seja representativo de um cenário aeronáutico, foi feita a adição de um ruído aleatório no nível de sinal recebido. Para isso foi escolhida a função de densidade de probabilidade adaptada de uma função "Cauchy", a qual pode ser facilmente implementada e conformada, possibilitando a ocorrência de diversos níveis. Esta estratégia permitiu estimular e excitar o programa controlador, simulando um cenário realístico.

4 Equipamentos Utilizados

4.1 Microcomputador

O programa controlador do PS foi instalado em um microcomputador PC Athlon 2800+ com 1 GB de memória RAM, 120 GB de HD SATA, monitor de 17" tela plana, placa-mãe Soyo Dragon Plus com chipset KT-600, placa de vídeo Nvidia 5200 com 128 MB de RAM. O sistema operacional utilizado foi o *Windows XP Professional*. Entretanto, uma configuração mais simples poderia igualmente ter sido empregada com sucesso, desde que o sistema não estivesse executando tarefas concorrentes que comprometessem o desempenho em tempo-real requerido pelo controlador, principalmente devido às telas gráficas. Neste PC foram instaladas 2 portas seriais adicionais, uma para o controle do PS e outra para o controle do receptor GPS, configuradas inicialmente como:

Equipamento	Porta	Taxa (bps)	Bit de paridade	Bits de dados	Stop-bit
Receptor GPS	COM1	38400	Nenhum	8	1
PS	COM2	9600	Nenhum	8	1

Tabela 2: Configuração da porta serial do Receptor GPS e do PS

4.2 Microcomputador Portátil

Os ensaios com o computador a bordo da aeronave empregaram o programa controlador de potência instalado em um Laptop Dell Inspiron 7500, de configuração bem mais modesta que o de mesa descrito anteriormente. Trata-se de um processador Intel Pentium III de 600 MHz, com 256 MB de RAM. Foi necessário utilizar um conversor de USB para serial, disponibilizando assim duas interfaces RS-232 requeridas pelo sistema de controle, conforme configuração descrita na Tabela 2.

4.3 Receptores GPS

Em geral, qualquer modelo de receptor GPS poderia ser utilizado. Os requisitos mínimos eram: possuir uma interface serial RS-232 para comunicação de dados e permitir o envio de mensagens do protocolo NMEA, informando, sobretudo o nível de sinal recebido. Nesta aplicação, em função da disponibilidade para testes e desenvolvimento do aplicativo, foram utilizados os receptores Ashtech Z-12 (Figura 4) e Z-FX (Figura 5) que são de propriedade da AEV. Estes receptores possuem 12 canais. Considerando que numa aplicação típica são rastreados até 11 satélites GPS [Parkinson e Spilker, 1996], estes receptores são perfeitamente adequados para a utilização com um PS. Dentre os dois receptores, o Ashtech Z-FX é o melhor por apresentar melhores características gerais, maior taxa de aquisição de dados (até 10/s) e maior faixa dinâmica.
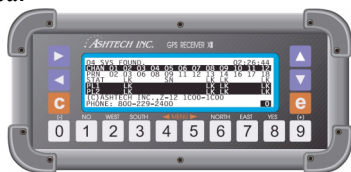
Figura 4: Receptor Ashtech Z-12

Figura 5: Receptor Ashtech Z-FX

4.4 Rádio - Modem

O emprego do rádio-modem é fazer chegar dados seriais do receptor até o PS e, adicionalmente, fazer chegar comandos do PS até o receptor GPS.

A escolha do rádio-modem XSTREAM-PKG X09-009PKC-R foi por conveniência e disponibilidade, uma vez que a Divisão de Sistemas Bélicos (ASB) possui um par. A configuração foi feita para se adequar às taxas de transmissão de dados seriais necessárias (i.e., 38400 bps ao se comunicar com os receptores GPS e 9600 bps ao se comunicar com o PS).

Em função da falta de documentação técnica, pouco se sabe sobre os algoritmos internos dos rádios-modem. Desta forma, as características técnicas deste equipamento foram obtidas experimentalmente (tabela 3), na qual: o tempo ΔT_{TX} corresponde ao tempo em que o pacote começa a ser enviado pelo canal; o tempo ΔT_{RX_TOTAL} corresponde ao tempo total, quando o pacote termina de ser recebido; e o número de blocos corresponde ao número de partições que o equipamento impõe para permitir o tráfego *duplex* pelo mesmo canal de RF. Desta forma, pode-se

perceber que nenhum *byte* chega do outro lado antes de 47 ms. Acima de 70 *bytes*, o pacote é dividido em blocos de 64 *bytes*.

Bytes	@38400	@9600	ΔT_{TX}	ΔT_{RX_TOTAL}	Blocos
1	260,42 us	1.041,67 us	47,2 ms	47,5 ms	1
70	18,23 ms	72,92 ms	98,8 ms	118 ms	1
71	18,49 ms	73,96 ms	97,6 ms	118 ms	2
134	34,90 ms	139,58 ms	97,6 ms	179 ms	2
135	35,16 ms	140,63 ms	97,6 ms	180 ms	3
192	50,00 ms	200 ms	97,6 ms	241 ms	3
193	50,26 ms	201,04 ms	97,6 ms	338 ms	4
262	68,23 ms	272,92 ms	97,6 ms	406 ms	4
263	68,49 ms	273,96 ms	97,6 ms	406 ms	5

Tabela 3: Tempo de transmissão do rádio-modem

Um dos maiores entraves ocorridos durante a fase de desenvolvimento do aplicativo e dos ensaios com o PS, foram os rádios-modem utilizados para realimentar dados ao receptor. Primeiramente, limitações em suas bandas passantes e em seus tempos de latência, potencializados por sua característica *half-duplex*, impuseram severas restrições ao aplicativo, exigindo que este último se adequasse àqueles. Esse tipo de problema veio à tona quando se empregou o receptor Z-FX, com intervalos de leitura inferiores a 500 ms.

Ao embarcar o programa controlador na aeronave (veículo), o tamanho do pacote de dados realimentados ao PS diminui drasticamente de, respectivamente, 33 a 226 bytes para apenas 3 *bytes*. O comando do nível de potência do PS, mostrado a seguir na Figura 6, requer 3 *bytes* (mais respectivos *start-bits* e *stop-bits*), com taxa de 9.600 bps.

5 Protocolo de Comunicação

5.1 Algumas instruções utilizadas

A taxa de comunicação de dados de 38400 bps com os receptores GPS foi escolhida por atender à demanda de dados nos casos mais restritivos, de tamanho do pacote e da taxa de aquisição selecionada. Inicialmente, quando se pretendia utilizar equipamentos de telemetria da AEV-T, esta seria a máxima taxa disponível, desta forma foi mantida a compatibilidade caso alguma migração entre sistemas ocorresse.

De acordo com o manual do receptor GPS Ashtech, alguns comandos (instruções) foram enviados ao receptor e outros recebidos do mesmo, através de suas portas seriais RS-232. Durante os ensaios, os receptores GPS embarcados foram programados mediante comandos do padrão NMEA, habilitando-os a enviar automática e periodicamente, as informações relativas a níveis de sinais recebidos, posição, velocidade, tempo, geometria e estado. Essa programação foi feita por intermédio do programa controlador de potência. Uma vez programado, o receptor GPS memoriza o último conjunto de comandos recebidos, ainda que sua alimentação seja descontinuada.

As principais mensagens NMEA recebidas pelo controlador dos receptores GPS foram: GSN e POS. A mensagem GSN devolve as informações referentes à intensidade de sinal recebido de cada satélite. Esta resposta poderá conter de 25 a 113 caracteres ASCII, em função de haver de 1 a 12 satélites recebidos e habilitados. A mensagem POS fornece diversas informações referentes à posição da antena do receptor, quais sejam: modo como a posição foi computada, número de

satélites usados no cômputo, tempo UTC em que a posição foi calculada, latitude, longitude, projeção verdadeira sobre o solo em graus, componente de velocidade relativa ao solo, componente de velocidade na vertical local, diluição de precisão no cômputo da posição (PDOP), diluição de precisão na determinação das componentes horizontais (HDOP), diluição de precisão na determinação da componente vertical (VDOP), diluição de precisão temporal (TDOP) e identificação da versão do *firmware* empregado. A resposta típica do comando POS contém 113 caracteres ASCII, considerando-se a recepção de 4 ou mais satélites.

5.2 Detalhes do Protocolo de Comunicação

A comunicação entre o Microcomputador PC e o PS foi feita por meio de interface serial assíncrona, saindo informações do PC via RS-232 padrão. Para isso bastam três linhas para que se estabeleça esta comunicação, quais sejam: TX, RX e GND. Estando o PS fisicamente localizado sobre a torre adjacente ao prédio X-30, o comprimento do cabo de comunicação serial excedeu o máximo suportado pelo padrão não balanceado RS-232. Sendo assim, o PS dispõe de duas interfaces de comunicação serial, sendo uma RS-232 e uma RS-422. Um adaptador especial foi colocado logo na saída RS-232 do PC, convertendo este padrão no padrão balanceado RS-422. O número de *bytes* por pacote é igual a três, ou seja, a cada atualização de potência o PC enviará uma seqüência composta por: *Start-Bit* + 8 *bits* de comando + *Stop-Bit* + *Star-Bit* + 8 *bits* processados + *Stop-Bit* + *Start-Bit* + 8 *bits* processados + *Stop-Bit*. Uma seqüência como esta é enviada ao PS de 1/s a 10/s, qual seja a taxa de saída de dados do receptor GPS à estação de telemetria. A Figura 6 mostra um exemplo da referida seqüência. Os oito *bits* de dados estão representados por traços ao centro do trem de pulsos, estando 4 deles ao final do segundo *byte* e os outros 4 no começo do terceiro *byte*. Note que o MSB está no segundo *byte*. Esta formatação se deve ao emprego de um Conversor Digital/Analógico serial no PS que exige uma seqüência derivada da mostrada.
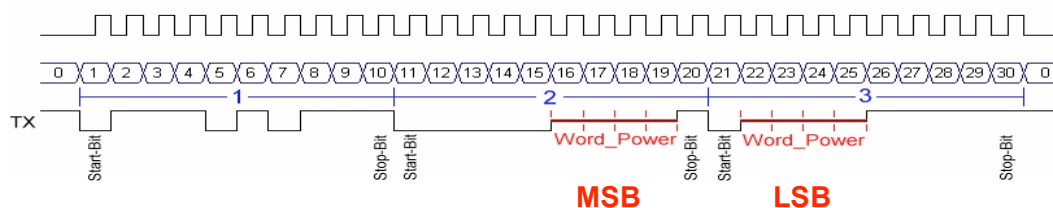
Figura 6: - Seqüência de bits a ser enviada pelo PC ao PS mediante interface de comunicação serial RS–232. A palavra de potência (os oito bits reservados em vermelho) são os únicos que sofrem mudança dentro da seqüência mostrada. A duração dos trinta bits consecutivos a 9600 bps é de 3,125 ms.

O primeiro *byte* corresponde à seqüência binária 11101011 (D7H ou ASCII x ou Î conforme a tabela ASCII, LSB primeiro). Esta é uma palavra de comando programada que informa ao PS que a presente comunicação corresponde de fato a uma ordem de atualização da potência transmitida, devendo, portanto, ser obedecida pelo PS. Outras seqüências com outros preâmbulos são ignoradas. Esta palavra de comando poderá ser modificada ou mesmo eliminada do protocolo caso necessário.

6 Interface com o operador

O programa controlador apresenta diferentes telas por onde o operador pode interagir. Uma vez iniciado o programa, aparecerá a primeira tela (figura 7) que permite a configuração das portas de comunicação seriais do receptor GPS e do PS. Os valores de configuração padrão são apresentados nesta figura (ou na Tabela 2 já mostrada). Além disto, pode-se também configurar o

código PRN empregado pelo PS (e.g., PRN #32) e as coordenadas da antena transmissora no formato latitude, longitude e altitude.

Pode-se optar pelo emprego do receptor Ashtech Z-FX ou Z12, nos modos “Solo” ou “Embarcado”. A modalidade embarcada foi empregada durante as fases de desenvolvimento, onde o programa foi instalado em um computador tipo Laptop que ficaria dentro da aeronave, junto ao receptor. Nesta configuração o rádio-modem envia ao PS os comandos de atualização de potência, uma vez que toda a informação do receptor é passada diretamente ao controlador através da porta serial RS-232.

A segunda tela do programa (figura 8), apresenta valores iniciais referentes ao receptor selecionado (neste caso, o Z-FX).
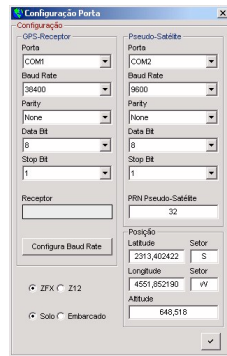
Figura 7: Tela inicial do programa controlador, mostrando configurações iniciais
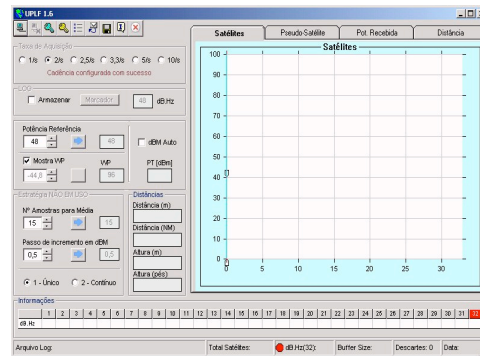
Figura 8: Tela principal do programa controlador

Alguns dos recursos e informações disponíveis ao operador são:

- Possibilidade de armazenamento das informações em arquivo (LOG), com ponto de marcação.
- Iniciar e parar a aquisição de dados
- Definição da taxa de aquisição do receptor em 1/s, 2/s, 2,5/s, 3,3/s, 5/s ou 10/s
- Re-início do receptor GPS, a fim de garantir as condições iniciais conhecidas e desejadas, inclusive referentes à taxa de comunicação.
- Armazenamento das configurações atuais ao receptor GPS, com a finalidade de se assegurar que tais configurações retornem da próxima vez que o receptor for ligado.
- Visualização e controle do nível de potência do PS a ser comandado, podendo utilizar o algoritmo desenvolvido no aplicativo para controle automático da realimentação do PS, ou o uso de controle de potência manual.
- Visualização e controle da WP e das estratégias de controle implementadas no aplicativo, podendo ser pela média das últimas amostras recebidas ou por passo de incremento/decremento, além do controle dos limites absolutos e relativos.
- Seleção de satélites, de forma automática ou manual, para os canais do receptor, com o máximo de 32. Na opção automática, são habilitados os 11 satélites com nível de potência mais alto, além do PS.
- Visualização da distância e altura do equipamento embarcado

- Visualização gráfica em tempo-real dos satélites selecionados, do pseudo-satélite, da potência recebida ou da distância percorrida e trajetória.

No rodapé da figura 8 encontra-se uma barra de estado, contendo: nome e caminho do arquivo LOG, número total de satélites em vista, indicação visual de captura (verde) ou não (vermelho) do PS, tamanho do bloco de memória temporário, utilizado na transferência de dados vindos do receptor, número de pacotes de dados corrompidos e descartados e data atual.

6.1 Telas gráficas para monitoração do controlador

O programa controlador dispõe de 4 telas gráficas atualizadas sob a mesma taxa de aquisição do receptor GPS. A primeira tela, intitulada “Satélites”, possui no eixo das abscissas o número dos PRN’s associados aos satélites (i.e., 1 a 32) e no eixo das ordenadas o valor do índice *C/N0* em dB·Hz dos satélites capturados. Os limites de escala do eixo y podem ser alterados pelo operador, pois a faixa dinâmica do receptor Z-12 é diferente do Z-FX. A presença de satélites em vista é mostrada através dos respectivos pontos verdes, (figura 9). O ponto vermelho corresponde ao PS (i.e., PRN#32). A cor de fundo do gráfico indica de forma rápida e visual o estado do controlador.

A segunda tela (figura 10), intitulada “Pseudo-Satélite”, apresenta as 20 últimas amostras do *C/N0* (dB·Hz) do PS. Além disto é possível visualizar simultaneamente a *WP* neste gráfico, desde que o limite superior da escala do eixo y seja adequadamente aumentado. Desta forma torna-se tangível a visualização da atuação do controle. Por outro lado, a presença de congelamentos ou perda de captura do sinal do PS fica facilmente caracterizada nesta tela, na qual o traço superior, em verde, corresponde à *WP* e o inferior, em vermelho, ao índice *C/N0*.
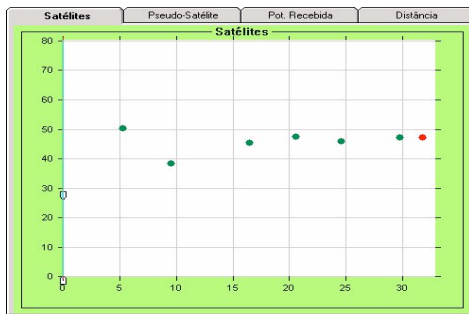
Figura 9: Tela indicativa dos satélites em vista com respectivos índices C/N0.
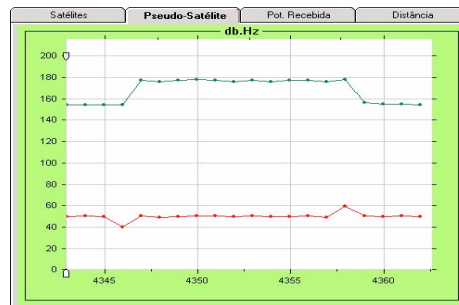
Figura 10: Tela indicativa de captura do pseudo-satélite versus últimas 20 amostras.

A terceira tela, intitulada “Pot. Recebida”, corresponde à variação da potência recebida em relação à potência de referência (P_{REF}), em dB·Hz. Nesta tela é possível alterar a escala do eixo y para ver em maiores detalhes o comportamento do controle de potência. Esta tela permite acompanhar quão fora está a potência recebida do valor ideal almejado (figura 11)

A quarta tela, intitulada “Distância” (figura 12), corresponde à representação bidimensional da posição do receptor no sistema de coordenada Leste-Norte-Acima (ENU) [Chatfield, 1997]. A origem corresponde ao centro de fase da antena transmissora. O traço no sentido noroeste-sudeste corresponde à pista do aeroporto de São José dos Campos (SBSJ), cujas extremidades correspondem às cabeceiras “15” e “33” respectivamente. As distâncias estão em km. Quando em operação, a posição do usuário é mostrada no gráfico em forma de um pequeno ponto que se movimenta, podendo deixar, caso habilitado, o rastro que indica a trajetória recente da aeronave. As componentes da distância em relação à origem são mostradas em campos específicos, bem como a distância radial, em metros.
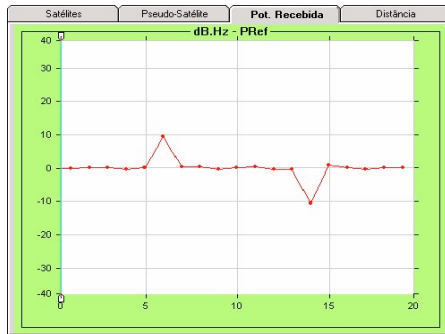
Figura 11: Tela indicativa de variação da potência recebida em dB·Hz em relação a PREF versus últimas 20 amostras, relativa ao sinal do PS.
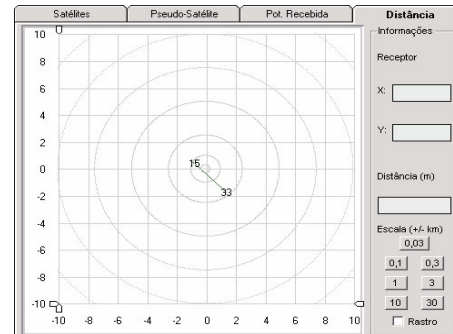
Figura 12: Tela com representação bidimensional de coordenadas ENU.

7 Ensaios

Os ensaios ocorreram juntamente às instalações da AEV, utilizando o helicóptero CH-55 e adjacências da pista do aeroporto de São José dos Campos. A primeira fase dos ensaios utilizou o enlace mostrado na Figura 2 com o receptor GPS e um rádio-modem, embarcados num carro, percorrendo parte da cidade de São José dos Campos e do CTA, a fim de aferir e desenvolver o aplicativo controlador. Na segunda fase dos ensaios foram previstos 5 vôos de ensaios em dois dias, mas apenas 4 foram executados. No quinto vôo a aeronave deveria se aproximar da base de testes, porém, por motivos operacionais e de segurança, este vôo não pode ser realizado. Nos demais ensaios (vôos), foram utilizados os receptores Ashtech Z-12 e Z-FX, e cada vôo teve, em média, a duração de 40 minutos, que foram fundamentais para a definição da relação *near/far*.

Na realização dos vôos, ocorreram dois tipos de falhas que se destacaram, prejudicando os resultados gerais dos ensaios. A primeira delas (mais grave) diz respeito ao mau funcionamento do rádio-modem instalado no helicóptero e a segunda é em relação às falhas operacionais, sobretudo pelo fator humano. As falhas operacionais foram mais pronunciadas nos vôos #3 e #4, onde o sincronismo entre as equipes de solo e de ar era mais necessário. As conclusões foram tomadas diante das informações armazenadas no arquivo LOG.
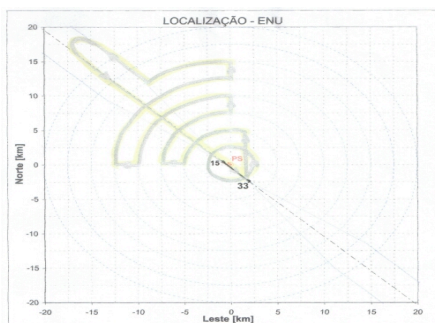
Figura 13: Projeção bidimensional do trajeto a ser seguido do vôo #4.
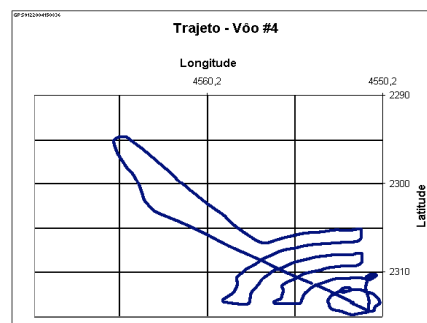
Figura 14: Projeção bidimensional do trajeto percorrido do vôo #4, com informações capturadas pelo aplicativo.

8 Conclusões

Considerando a complexidade dos ensaios realizados, nossa avaliação, como esperada, foi muito positiva. Conseguiu-se receber um sinal GPS transmitido por um PS, desde menos de 20 metros até cerca de 40 km, perfazendo uma relação *near/far* superior a 1:2000. Obviamente há que se considerar o problema de interferências com não-participantes. A recepção do PS por uma antena apontada para cima também foi muito importante no resultado. Trata-se de um cenário bastante desfavorável, onde a variação de ganho com a elevação sofre muita modificação. Graças à técnica RU e às estratégias de recuperação e controle implantadas no aplicativo UF, isto foi especialmente possível em tão ampla gama de distâncias e de dinâmicas de aeronaves.

Como todo aplicativo, este programa pode e deve ser aperfeiçoado, mediante sintonias finas em seus algoritmos e coeficientes, otimizações diversas em sua estrutura e aspectos de apresentação. Entende-se, entretanto, que da forma em que está, permite a operação em parceria com o PS desenvolvido e com os receptores GPS empregados, atendendo aos objetivos de coadjuvante a que foi proposto.

O PS desenvolvido no ITA, inicialmente concebido para aplicações aeronáuticas, pode facilmente se adequar a aplicações em Agricultura de Precisão. Descreveu-se, em linhas gerais, o programa controlador de potência elaborado com a finalidade de atender à proposta de RU.

9 Agradecimentos

Agradeço a todos que participaram e contribuíram para a realização dos testes e desenvolvimento do aplicativo no projeto Pseudo-Satélite, de maneira especial, à ASB, ao ITA, ao Sr. Nelson Paiva Oliveira Leite, ao Sr. Jorge Tadano e à Divisão de Ensaios em Vôo (AEV) do IAE-CTA e seus servidores.

Referências

COBB, H. S.: GPS Pseudolites: *Theory, Design, and Applications*, Ph.D. dissertation, Dept. of Aeronautics and Astronautics, Stanford University, California, USA, 1997.

Klein, D.; Parkinson, B. W.: *The Use of Pseudo-Satellites for Improving GPS Performance, in the Global Positioning System*, Vol. 3, Institute of Navigation, Washington, DC, USA, 1984.

ZANDONADI Jr., D.; Walter, F.: Pseudo-Satélite – Transmissor/Codificador de Sinais GPS para Aplicações GBAS: Descrição, Projeto e Implementação, IX ENCITA, 7-9 de Outubro de 2003, São José dos Campos, SP

ARINC Research Corporation, ICD-GPS-200, *Navstar GPS Space Segment/Navigation User Interfaces*, Revision C, 10 October 1993

ZANDONADI Jr., D.; Superação do problema near/far em pseudo-satélites GPS mediante técnica de realimentação do receptor do usuário. 2005. 298f. Tese (Doutorado em Telecomunicações) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

FAA, *Criteria for Approval of Category III Landing Weather Minima (AC-120-28C)*. U.S. Government Printing Office, Washington, DC, 1984.

Chatfield, Averil B., *Fundamentals of High Accuracy Inertial Navigation*, Vol. 174, American Institute of Aeronautics and Astronautics, 1997

Parkinson, B. W.; Spilker Jr., J. J.: *Global Positioning System: Theory and Applications* Progress in Aeronautics and Astronautics Vol. 163, AIAA, 1996.

Utilização de Máquinas Virtuais para Implantar um Mecanismo Transparente de Detecção de Intrusão em Servidores Web

Luciano Raitz (FURB/Especialização)
mrraitz@yahoo.com.br

Francisco Adell Péricas, Ms (FURB/DSC)
pericas@furb.br

Resumo. Este artigo apresenta uma arquitetura confiável para uso de detectores de intrusão através da utilização de máquinas virtuais, que, com as vantagens de portabilidade e custo, vêm se destacando para a criação de servidores. A arquitetura proposta faz uso de máquinas virtuais para deixar o sistema de detecção de intrusão invisível e inacessível, caso haja uma invasão. Os testes apresentados mostram que a utilização desta arquitetura é funcional e viável para uma arquitetura de servidores.

Palavras-chave: Máquinas virtuais; *Firewall*; IDS; Servidores Web.

1 Introdução

A Web enfrenta diferentes ameaças desde a sua criação, que aumentam com o passar dos tempos. Com a necessidade de criação de novas funcionalidades em um ambiente em crescimento, projetistas podem não ter dado a devida atenção para a segurança dos sistemas desenvolvidos.

As redes de computadores têm crescido constantemente, acompanhadas em uma mesma proporção pela área de segurança, para manter a integridade de seus serviços disponibilizados. As preocupações com as questões relativas à segurança são reais e não podem ser deixadas de lado, pois existem pessoas (*cracker*) com intenção de invadir, prejudicando e bisbilhotando sites com conteúdo confidencial. No entanto, métodos para bloquear esses intrusos estão sendo desenvolvidos, fazendo com que fique cada vez mais difícil a ação deles.

Diversas ferramentas contribuem para aumentar a segurança de um sistema de computação, entre as quais se destacam os *Firewalls* e os sistemas de detecção de intrusão (IDS – *Intrusion Detection System*). Tais sistemas monitoram continuamente a atividade de um ambiente computacional, buscando evidências de intrusões (LAUREANO, 2004). Estas ferramentas fazem a análise automatizada das informações, viabilizando auditorias do sistema.

Máquinas virtuais, segundo Chen e Noble (2001), podem ser usadas para incrementar a segurança de um sistema computacional contra ataques aos seus serviços. O uso de máquinas virtuais vem se tornando interessante também em sistemas computacionais modernos, devido a suas vantagens em termos de custo e portabilidade (BLUNDEN, 2002). A utilização de máquinas virtuais evita a necessidade de se ter um equipamento para cada servidor de sua rede: pode-se ter um único equipamento e neste várias máquinas virtuais com os diversos aplicativos.

Devido à vulnerabilidade intrínseca dos sistemas de detecção de intrusão, que podem ter seu processo interrompido sem maiores dificuldades caso haja uma invasão no servidor onde estiver rodando, tem-se a necessidade de aumentar o nível de segurança destes sistemas. Para que se tenha um aumento do nível de segurança em servidores Web, este trabalho propõe uma solução onde serão utilizadas duas ferramentas em conjunto: a primeira, uma máquina virtual (como um sistema convidado) que será preparada para ser o servidor; a segunda um sistema de detecção de intrusão, que deve estar configurado de tal maneira que o invasor não tenha acesso e não tenha como ocultar

os rastos da sua invasão, até mesmo porque ele sequer tem como saber da sua presença. No caso de uma invasão, a máquina virtual representa uma armadilha de rede (*honeynet*).

Magalhaes (2004) escreveu um artigo que serve como referência na criação de armadilhas de rede, destacando o objetivo de empregá-las nas redes de empresas. Essas armadilhas de redes são usadas como uma estratégia para acompanhar e aprender estratégias e conhecer ferramentas utilizadas em ataques de redes por *hackers* e uma das formas de implementá-las é através do uso de máquinas virtuais. Outro artigo que aborda as *honeynets* implementadas através de máquinas virtuais é de autoria de Clark (2001) e está disponível no site do SecurityFocus.

O presente artigo apresenta o resultado de um trabalho de Pós-Graduação que teve como objetivo desenvolver uma implantação de uma arquitetura de detecção de intrusão, utilizando os recursos de máquinas virtuais como servidores de aplicações para a internet, e apresentar os benefícios dessa arquitetura em relação à segurança de servidores Web, ao ocultar tanto os softwares de monitoração como os registros gerados.

O aumento do nível de segurança para aplicações web é uma necessidade, tendo em vista que o crescimento da rede internet é inevitável e, portanto, sua vulnerabilidade também. Mecanismos de monitoração do tráfego de dados de acesso a aplicativos web são fundamentais, mas tanto as aplicações quanto seus relatórios são suscetíveis ao acesso por parte de *hackers*.

2 Segurança de servidores web

Existem várias ferramentas para a proteção de um servidor e da sua própria rede. Não existe um único software ou hardware que realiza todo o trabalho de proteção. Normalmente os sistemas para proteger uma rede são vários e servem para complementar um ao outro. Exemplos de ferramentas para a proteção são o *Firewall* e o IDS.

As ferramentas para segurança de computadores e redes são necessárias para proporcionar transações seguras. Geralmente, as instituições concentram suas defesas em ferramentas preventivas como *Firewalls*, mas acabam ignorando as ferramentas de detecção de intrusão.

O *Firewall* é o mecanismo de segurança interposto entre a rede interna e a rede externa com a finalidade de liberar ou bloquear o acesso de computadores remotos aos serviços que são oferecidos em um perímetro ou dentro da rede corporativa. Este mecanismo de segurança pode ser baseado em hardware, software ou uma mistura dos dois.

A construção de um *Firewall* é raramente constituída de uma única técnica. É, ao contrário, um conjunto balanceado de diferentes técnicas para resolver diferentes problemas. O objetivo de qualquer *Firewall* é criar um perímetro de defesa projetado para proteger os recursos internos de uma organização.

Atualmente um *Firewall* não garante mais que a empresa esteja livre de sofrer ataques. Outra ferramenta que se destaca é o Sistema de Detecção de Intrusão (IDS), uma ferramenta que visa auxiliar as empresas a proteger sua rede contra ataques e invasões.

Uma forma mais avançada de segurança combina o IDS com o *Firewall*, onde o IDS detecta o intruso e interage com o *Firewall* para que o tráfego de futuros pacotes possa ser negado.

3 Máquinas virtuais

As máquinas virtuais foram originalmente desenvolvidas para centralizar os sistemas de computador utilizados no ambiente VM/370 da IBM. Naquele sistema, cada máquina virtual simulava uma réplica física da máquina real e os usuários tinham a ilusão de que o sistema estava disponível para seu uso exclusivo (SUGERMAN; GANESCH; BENG-HONG, 2001).

Segundo Campos (2003), pode-se definir uma máquina virtual (VM) como uma máquina abstrata, ao contrário de uma máquina emulada, que permite que a máquina real seja particionada de tal modo que diversos sistemas operacionais sejam executados ao mesmo tempo.

Um emulador é um software que simula um computador real. Um emulador "engana", fazendo com que todas as operações da máquina real sejam implementadas em um software. Isso possibilita executar um aplicativo de uma plataforma em outra, por exemplo, um aplicativo do Windows executando no Linux. Devido à simulação quase que total das instruções de um computador, um emulador perde muito em eficiência ao traduzir cada instrução da máquina real.

Segundo Laureano (2004), a funcionalidade e o nível de abstração de uma máquina virtual encontra-se em uma posição intermediária entre uma máquina real e um emulador, de forma que os recursos de hardware e de controle são abstraídos e usados pelas aplicações.

O software de máquina virtual cria um ambiente através de um monitor de máquina virtual (*Virtual Machine Monitor* – VMM), que é um computador com seu próprio sistema operacional dentro de outro sistema operacional (*host*). Este monitor pode criar várias máquinas virtuais sem que nenhuma interfira na outra e também não interfira no sistema real onde o software de máquina virtual está instalado. A máquina virtual pode ser uma cópia do hardware da máquina real, fazendo com que o sistema operacional na máquina virtual pareça estar executando diretamente sobre um computador real.

Cada máquina virtual trabalha como um PC completo, com direito até a BIOS e configuração do *Setup*. Dispositivos como o CD-ROM e unidades de disquetes podem ser compartilhados entre as máquinas virtuais e o sistema *host*, em alguns casos até mesmo simultaneamente (uma unidade de CD pode ser acessada em todos os sistemas).

De acordo com Rosenblum (2004), embora as funcionalidades dos diversos ambientes virtuais sejam diferentes, todos compartilham de atributos comuns como:

- compatibilidade do software: a máquina virtual fornece uma abstração compatível de modo que todo o software escrito para ela funcione;
- isolamento: faz com que os softwares que funcionam na máquina virtual e nas outras máquinas virtuais e máquinas reais estejam totalmente isolados;
- encapsulamento: é usado para manipular e controlar a execução do software na máquina virtual;
- desempenho: adicionar uma camada de software a um sistema pode afetar o desempenho do software que funciona na máquina virtual, mas os benefícios de sistemas virtuais compensam.

3.1 Uso de máquinas virtuais

Ao longo dos anos, as máquinas virtuais vêm sendo utilizadas com vários fins, como processamento distribuído e segurança. Um uso freqüente de sistemas baseados em máquinas virtuais é a chamada “consolidação de servidores”: em vez da utilização de vários equipamentos com seus respectivos sistemas operacionais, utiliza-se somente um computador, com máquinas virtuais abrigando os vários sistemas operacionais e suas respectivas aplicações e serviços.

Segundo Sugerman, Ganesh e Beng-Hong (2001), muitos dos benefícios das máquinas virtuais utilizadas no ambiente dos *mainframes* também podem ser obtidos nos computadores pessoais. Algumas vantagens para a utilização de máquinas virtuais em sistemas de computação são:

- podem simular a existência de várias placas de rede dentro da máquina virtual;

- pode-se ter várias máquinas virtuais cada uma com um sistema operacional diferente sem ter que particionar o disco rígido;
- facilita o aperfeiçoamento e testes de novos sistemas operacionais;
- auxilia no ensino prático de sistemas operacionais e programação ao permitir a execução de vários sistemas para comparação no mesmo equipamento;
- simula configurações e situações diferentes do mundo real, como por exemplo, mais memória disponível ou a presença de outros dispositivos de E/S;
- simula alterações e falhas no hardware;
- garante a portabilidade das aplicações;
- permite o desenvolvimento de novas aplicações para diversas plataformas, garantindo a portabilidade destas aplicações;
- diminuição de custos com hardware, através da consolidação de servidores;
- facilidades no gerenciamento, migração e replicação de computadores, aplicações ou sistemas operacionais;
- provê um serviço dedicado para um cliente específico com segurança e confiabilidade;
- assegura a compatibilidade e a migração de sistemas;
- facilidade para treinamentos e demonstrações de produtos.

Uma das principais desvantagens na utilização de máquinas virtuais fica por conta do desempenho, pois o custo para a execução de um processo fica mais alto que em um computador real. Num trabalho desenvolvido por Santos e Teodorowitsch (2004) há um estudo do desempenho de máquinas virtuais e um comparativo deste desempenho entre as principais máquinas virtuais existentes.

3.2 Exemplos de máquinas virtuais

Atualmente existe uma grande variedade de máquinas virtuais, tais como o *Java Virtual Machine*, o CLR do .NET da Microsoft, o VMware e o Virtual PC para Windows. Além destes, também existe um sistema operacional chamado *Virtual Machine* (VM), que roda em máquinas IBM *mainframes*.

3.2.1 Java Virtual Machine

Tendo sido originalmente concebida para o desenvolvimento de pequenos aplicativos e programas de controle de aparelhos eletrodomésticos e eletroeletrônicos, o Java mostrou-se ideal para ser usada na rede internet. O que o torna tão atraente é o fato de programas escritos em Java poderem ser executados virtualmente em qualquer plataforma, mas principalmente em Windows, Unix e Mac.

Um programa fonte escrito em Java é traduzido pelo compilador para os *bytecodes*, isto é, o código de máquina de um processador virtual, chamado *Java Virtual Machine* (JVM), que é o monitor da máquina virtual Java.

A JVM é um programa capaz de interpretar os *bytecodes* produzidos pelo compilador. Com isso, um programa Java pode ser executado em qualquer plataforma, desde que seja dotada de uma JVM. É o caso dos programas navegadores mais populares, como o Netscape Navigator e o Internet Explorer, que já vêm com uma JVM. A vantagem desta técnica é evidente: garantir uma maior portabilidade para os programas Java em códigos-fonte e compilados.

3.2.2 Framework .Net

O .Net, produto da Microsoft, também é uma máquina virtual, semelhante ao Java. Desenvolvido sobre os padrões de Web Services XML, o .Net possibilita que sistemas e aplicativos, novos ou já existentes, conectem seus dados e transações, independente do sistema operacional, tipo de computador ou de dispositivo móvel que sejam utilizados, ou de que linguagem de programação tenha sido utilizada na sua criação.

A *Common Language Infrastructure* (CLI) contém a especificação dos seus principais serviços. Ela implementa a tecnologia que permite que um aplicativo seja desenvolvido em diversas linguagens de programação e executado num mesmo ambiente de execução. Ela é suportada por diversos sistemas operacionais (sistemas Win32, FreeBSD, MAC OS X e em fase de migração, o Linux), o que é de fundamental importância em aplicações distribuídas, como os sistemas multiagentes.

Tudo isso é possível porque o *Common Language Runtime* (CLR) permite e fornece sistemas de tipos comuns para todas as linguagens baseadas no .Net Framework.

3.2.3 VMware

O VMware é uma máquina virtual que emula um PC baseado em Intel e consegue, portanto, passar muitas instruções diretamente para a CPU para execução, sem tradução de intermediários, aumentando assim a velocidade de processamento. Isto é um pouco diferente de uma JVM, onde, por exemplo, o emulador tem de traduzir o código *bytecode* de Java para instruções Intel, antes de executá-las.

Os arquivos são armazenados em "discos virtuais" que aparecem como arquivos dentro de uma pasta no sistema *host* e cada sistema operacional pode ter uma configuração de rede distinta, com seu próprio endereço IP e tudo mais. As máquinas virtuais ficam acessíveis na rede, como se fossem realmente PCs completos, permitindo que se rode um servidor Web ou um programa P2P dentro de uma máquina virtual, sem comprometer a segurança do sistema principal.

O VMWare é um produto comercial, destinado principalmente a servidores. É muito usado em provedores de acesso que podem rodar várias máquinas virtuais dentro de um mesmo servidor e assim oferecer *hosts* "semi-dedicados" a um custo bem mais baixo que o de servidores realmente exclusivos. O cliente continua tendo acesso completo a seu "servidor", apenas o desempenho pode ser menor, de acordo com o número de máquinas virtuais por *host*.

Por razões de desempenho, o monitor do VMware utiliza uma abordagem híbrida para implementar a interface do monitor com as VM's. O controle de exceção e gerenciamento de memória é realizado através da manipulação direta do hardware, mas para simplificar o monitor, o controle de E/S é do sistema anfitrião. Através do uso de abstrações para suportar a E/S, o monitor evita manter *device drivers*, algo que os sistemas operacionais já implementam adequadamente.

3.2.4 Microsoft Virtual PC

O Microsoft Virtual PC 2004, também conhecido como VirtualPC antes de ser adquirido da Connectix pela Microsoft. Esta máquina virtual é muito parecida com o VMware. Pode ser instalado e configurado na maioria dos sistemas operacionais baseados em Intel, sem a necessidade de *drivers* específicos.

Algumas características são: suporte para até quatro adaptadores de rede por máquina; configuração baseada na linguagem XML para facilitar a cópia da estação virtual; e suporte para até 4 GB de memória.

4 Arquitetura para proteção de detectores de intrusão

Existem problemas com as informações geradas através do IDS de Rede (NIDS), já que os registros podem ser facilmente apagados ou alterados após um ataque bem sucedido. Essa fragilidade se dá porque, através de um ataque bem sucedido, o invasor pode conseguir uma sessão da estação e a partir desse momento, ele pode fazer seu ataque no servidor invadido ou apenas utilizar a máquina para atacar outros servidores, evitando fazer ataques de sua própria estação. Porém, antes de encerrar sua sessão, ele pode ir até seu IDS e simplesmente apagar as informações que seriam capazes de informar as atividades feitas por ele através da estação invadida.

Outro problema é a recuperação da estação invadida, que em muitos casos tem que ser completamente instalada e re-configurada ou até mesmo ser recuperada de uma imagem que está em outra estação. Isso demanda tempo, e em sistemas críticos isso pode causar um prejuízo muito grande.

A arquitetura proposta por este trabalho, cria uma maneira segura para proteger os dados do NIDS através do uso de máquinas virtuais. O servidor a ser protegido deve ser executado na máquina virtual com toda sua configuração como se estivesse em um Hardware comum (inclusive com o seu NIDS). Mas, além de existir o NIDS no servidor instalado na máquina virtual (sistema convidado), existe também o mesmo NIDS instalado na máquina real (sistema *host*) a qual passa a ser um *host* e este está protegido, pois o sistema *host* deve ficar sem configurações de rede e desta forma inacessível pelo invasor e, além disso, oculto, pois não há como se saber de sua existência.

4.1 Desenvolvimento da proposta

Para a criação da arquitetura proposta, foi utilizado um hardware baseado na tecnologia Intel. A arquitetura criada é formada por:

- Windows XP Professional;
- GFI LANguard N.S.S.;
- IDS Snort;
- MySQL;
- Apache versão 2.0.48 (Win32);
- *Analysis Console for Intrusion Detection* (ACID);
- VMware Workstation Versão 4.5.1 build-7568.

Todas as ferramentas que fazem parte da arquitetura foram instaladas tanto no sistema *host* como no sistema convidado.

Conforme é visto na Figura 1, tanto o sistema *host* quanto o sistema convidado estão ativos. No VMware, além do sistema convidado que será utilizado para mostrar os testes, também existem outras duas máquinas virtuais, uma com o Windows XP Professional e outra com o Kurumin (Linux).

A máquina virtual foi criada usando o método de utilização de discos virtuais, para que possa ser facilitada a criação de *backups*. Pois desta maneira não há a necessidade de particionar o disco rígido, facilitando a cópia da máquina virtual já que o disco rígido do sistema convidado será apenas um arquivo no sistema *host*.
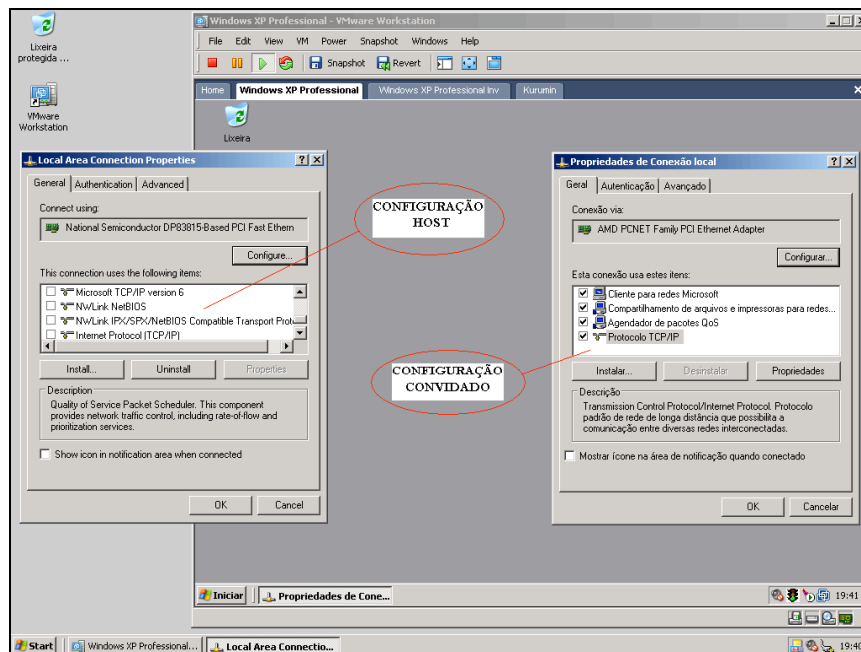
Figura 1: Configurações de rede

A Figura 1 mostra como deve ficar a configuração de rede do sistema *host* e do sistema convidado. Para o funcionamento correto da arquitetura proposta, os sistemas devem estar configurados de maneira que o sistema *host* não tenha protocolos (como TCP/IP, NetBios e IPX/SPX) ativos, tornando o sistema *host* inacessível pela rede. Já o sistema convidado deve ter configuração de rede completa com TCP/IP, ou seja, configurado de tal forma como se fosse um servidor comum.

Com o sistema *host* sem qualquer protocolo ativo, este fica inacessível de qualquer outra estação. Isso faz com que as atualizações do sistema *host* não possam ser realizadas pela rede de computadores, pois a máquina será inexistente na rede a qual o hardware está conectado.

Na etapa seguinte de montagem da arquitetura, foi instalado o IDS (Snort) tanto no sistema *host* quanto no sistema convidado, que foi configurado de tal maneira que os pacotes que devem ser capturados, somente sejam os que tenham endereço destino ou origem do sistema convidado. Essa configuração que determina quais pacotes devem ser capturados é informada no arquivo “snort.conf” que contém a configuração do Snort.

No Snort foi colocado o endereço IP dos pacotes que devem ser capturados, tanto os pacotes de entrada e saída. Essa mesma configuração foi utilizada tanto no sistema host, como no sistema convidado, para que ambos capturem os mesmos pacotes.

O MySQL foi instalado nos dois sistemas para que os pacotes capturados fossem gravados no banco de dados, e assim disponibilizando-os para futuras consultas para análise dessas informações.

Para a administração do IDS (Snort), pode ser utilizado o IDScenter, que além de servir como ferramenta para a análise dos dados, também pode ser utilizado para configuração das regras do IDS, sabendo que a cada regra criada no sistema host deve obrigatoriamente ser criada no sistema convidado e vice-versa.
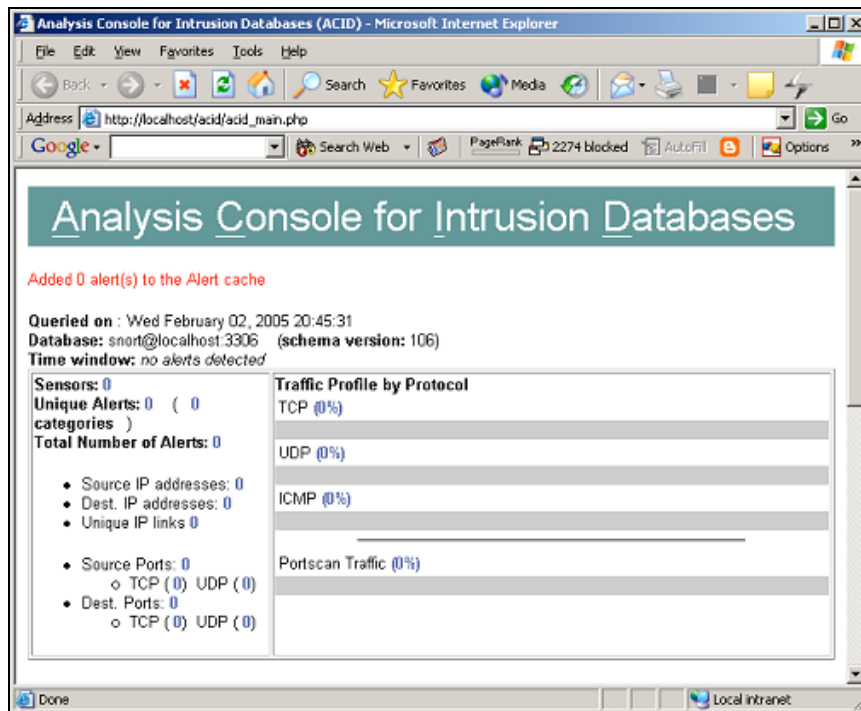
Figura 2: ACID

A Figura 2 mostra a tela principal do ACID, ferramenta utilizada para a consulta dos pacotes capturados, que apresenta as estatísticas gerais como o número de alertas divididas pelo protocolo e contador de portas de origem e de destino para regras disparadas. Clicando em cada uma dessas informações, é possível obter informações mais detalhadas, e cada pacote capturado pode ser exibido em um formato decodificado. O ACID também permite a criação de consultas com vários parâmetros no banco de dados, desde o tipo de assinatura até o conteúdo dos pacotes.

Para a realização de uma varredura de portas foi utilizado o software GFI LANguard N.S.S. Neste software, basta informar o IP que se deseja realizar o "*portscan*" e dar início ao processo. Dessa maneira, foi possível obter dados para a análise de captura do IDS para futura comparação.

4.2 Comprovação da inacessibilidade do IDS no *host*

Com os sistemas instalados e configurados, pode-se inicializar o Snort. Na Figura 3 observa-se a captura de pacotes dos dois sistemas, através de um comando do Snort executado no *prompt*, onde as duas máquinas, tanto o sistema *host* quanto o sistema convidado, capturam os mesmos pacotes.

O total de pacotes processados é grande, mas apenas parte desses pacotes foi armazenada, pois os outros pacotes foram descartados pelas regras de assinatura do Snort. Existe a possibilidade de todos os pacotes serem armazenados, bastando configurar o IDS em modo promíscuo, mas o volume de dados será grande, além de não ter a necessidade de avaliar todos os pacotes, já que muitos podem não representar ataques.
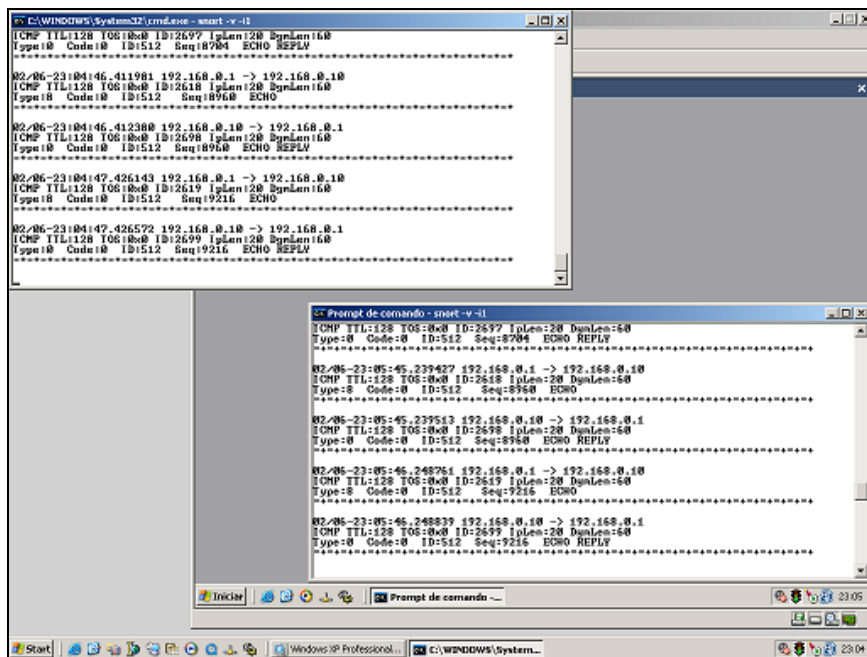
Figura 3: Capturando pacotes

Para que possa ser demonstrada a igualdade dos conjuntos de pacotes capturados, o Snort foi interrompido em ambos os sistemas para ser apresentado um resumo dos pacotes capturados. Os valores processados pelos sistemas *host* e convidado após um PortScan têm de ser os mesmos.
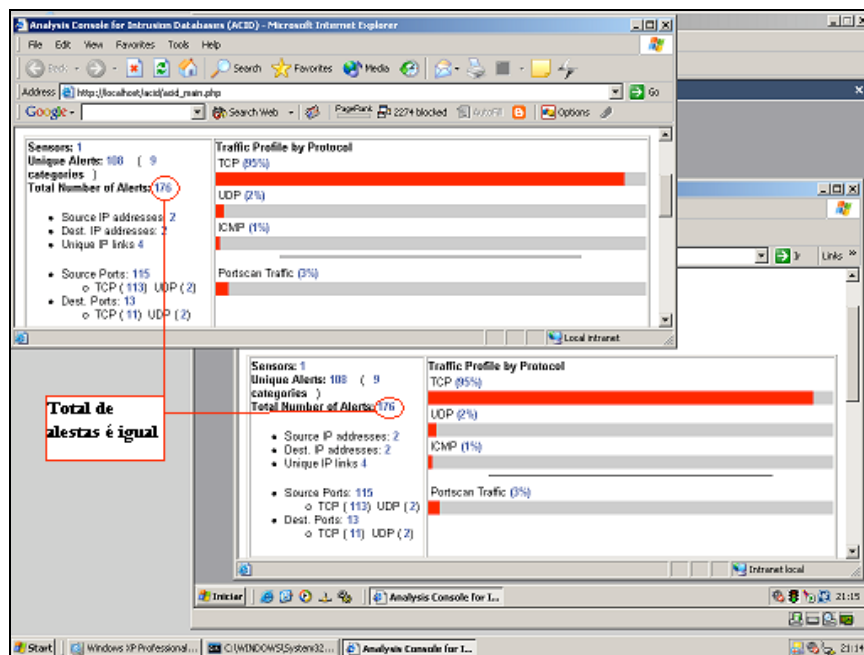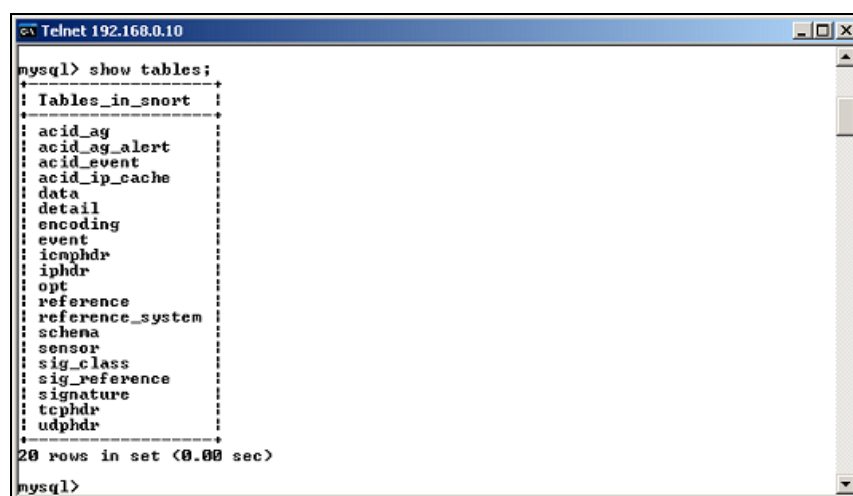
Figura 4: Resultados no ACID

Na Figura 4, através do ACID, pode-se observar o total de pacotes armazenados pelo Snort. Como pode ser observado, a maioria dos pacotes foi descartada, já que o Snort processou mais de 1500 pacotes e apenas 176 pacotes foram armazenados. É de fundamental importância que o IDS esteja configurado da mesma maneira no *host* como no sistema convidado (que as regras sejam iguais), caso contrário, a comparação do total de pacotes assim como os pacotes armazenados pelo IDS não teriam funcionalidade nenhuma.

Em um processo de invasão do sistema, podem ser utilizadas diversas ferramentas, como o uso de *scanners* para detectar quais portas estão abertas, *sniffers* para roubar informações como usuários e senhas e programas que procuram por vulnerabilidades conhecidas nos sistemas e que não tenham sido atualizadas.

Este trabalho não se propõe apresentar técnicas de invasão de um sistema nem ferramentas que podem ser utilizadas para isso.

Para demonstrar a funcionalidade da arquitetura, foi simulado um ataque, através de uma conexão *telnet* conforme Figura 5, onde o *cracker* invadiu o sistema convidado e acessou o MySQL listando as tabelas do Snort. A partir desse momento, qualquer informação pode ser alterada ou até mesmo eliminada do sistema para encobrir o seu rastro.

Figura 5: Acesso MySQL pelo *telnet*

A vantagem da utilização de máquinas virtuais para proteção do sistema, o invasor pode ter removido as informações sobre a sua conexão de *telnet* para encobrir o rastro deixado no sistema. Mas como o sistema estava sendo monitorado em dois lugares, o IDS do sistema *host* permanece com as informações sem nenhuma alteração. Isso pode ser observado na Figura 6 onde é mostrado o resultado dos pacotes no ACID: onde o IDS do *host* permanece com os 179 pacotes iniciais capturados e o sistema convidado, está com 178 pacotes já que o pacote *telnet* foi eliminado.
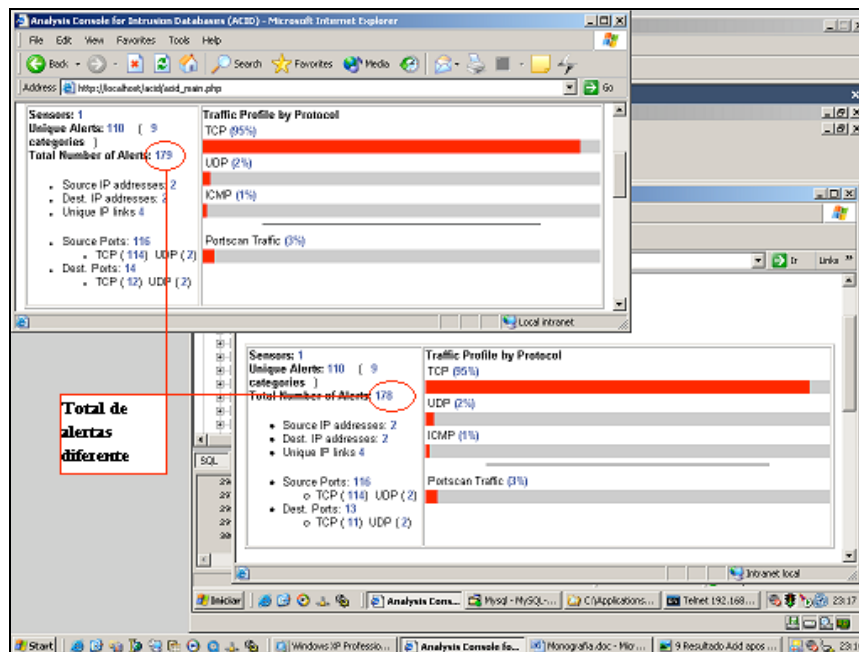
Figura 6: Resultado após ataque

Após um ataque, esta máquina virtual pode ser copiada para uma outra estação para que possa ser estudada a fim de observar as falhas em sua segurança e pode ser iniciado um backup da máquina virtual com a posição antes do ataque para colocar o sistema no ar novamente.

5 Conclusões

O objetivo do trabalho desenvolvido no curso de Pós-Graduação foi montar uma arquitetura para manter segura as informações de um IDS. Para que a arquitetura proposta pudesse ser montada, foram estudadas as ferramentas IDS e as máquinas virtuais.

Os *crackers* estão sempre desenvolvendo e procurando novas vulnerabilidades para atacar um computador, independente do sistema operacional que esteja sendo usado. As ferramentas de proteção acabam sendo um desafio a mais para ser quebrado pelos invasores, isso faz com que se busquem cada vez mais mecanismos para manter a integridade do sistema a ser protegido.

A instalação da ferramenta IDS é importante devido ao número de ataques que cresce constantemente. É importante poder detectar uma invasão a um sistema antes que se tenha algum tipo de prejuízo, o qual pode até arranhar a própria imagem da empresa. Mas os sistemas de detecção são vulneráveis aos invasores, pois os dados normalmente ficam armazenados em banco de dados e arquivos de registro que podem ser alterados pelo invasor.

Com a arquitetura proposta neste artigo o IDS ficou protegido, pois os dados de registro são armazenados em duplicidade, tanto no sistema *host* quanto no sistema convidado. Com isso, a arquitetura proposta mostrou ser segura, garantindo a integridade dos dados de registro armazenados. Essa arquitetura não diminui a possibilidade de um *cracker* invadir o sistema, mas, ao implementar um *honeynet*, possibilita detectar uma tentativa ou até mesmo uma invasão bem sucedida. Mesmo que o *cracker* remova os vestígios de que invadiu o sistema, ele estará apenas eliminando os rastros no sistema convidado e não no sistema *host*.

Com as informações armazenadas no sistema *host*, também poderão ser elaborados relatórios para dados estatísticos, podendo ser utilizados para uma melhor configuração do *Firewall* e do IDS, melhorando a detecção dos invasores devido ao aperfeiçoamento das políticas e planejamento da segurança. As informações armazenadas também servem para a reconstrução de um ataque ou na determinação de um ataque.

Além de a detecção poder ser realizada com sucesso, tem-se o benefício da recuperação do servidor convidado com uma configuração anterior ao ataque, bastando restaurar um backup da máquina virtual em seu estado anterior. E a máquina atacada pode ser iniciada em outro sistema *host* para posterior análise dos dados devido a portabilidade das máquinas virtuais.

É importante salientar que as ferramentas e a arquitetura utilizadas e apresentadas neste artigo não dispensam a utilização de *Firewall*, de IDS e de outras ferramentas de segurança, tais como antivírus e antispyware, para a proteção da rede e de seus computadores.

Referências

BLUNDEN, B. **Virtual Machine Design and Implementation in C/C++**. Plano, Texas – USA, 2002. Wordware Publishing.

CAMPOS, V. R. A. **Estudo comparativo de linguagens de programação**. Salvador, 2003. Disponível em: <http://twiki.im.ufba.br/pub/MAT052/RelacaoMonografias/monografia.doc>. Acesso em: 12 nov 2004.

CHEN, P. M.; NOBLE, B. D. **When Virtual is Better Than Real**. In: Proceedings of the 2001. Workshop on Hot Topics in Operating Systems (HOTOS).

CLARK, M. **Virtual Honeynets**. 2001. SecurityFocus. Disponível em: <http://www.securityfocus.com/infocus/1506>. Acesso em: 10 jun 2005.

LAUREANO, M. A. P. **Uma abordagem para a proteção de detectores de intrusão baseada em máquinas virtuais**. Dissertação (Mestrado em Informática Aplicada) - Centro de Ciências Exatas e de Tecnologia, Pontifícia Universidade Católica do Paraná, Curitiba, 2004. 103 f.

MAGALHAES, R. M. **Understanding Virtual Honeynets**. 2004. Disponível em: <http://www.windowsecurity.com/articles/Understanding_Virtual_Honeynets.html>. Acesso em: 10 jun 2005.

ROSENBLUM, M. **The Reincarnation of Virtual Machines**. Disponível em: <http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=168>. ACM Queue vol. 2, no. 5 - July/August 2004. Acesso em: jan 2005.

SANTOS, G. M.; TEODOROWITSCH, R. **Análise de Desempenho de Máquinas Virtuais**. Trabalho de Conclusão de Curso - Curso de Ciência da Computação, Universidade Luterana do Brasil (Ulbra), Gravataí-RS, 2004.

SUGERMAN, J.; GANESH, V.; BENG-HONG L. **Virtualizing I/O Devices on Vmware Workstation's Hosted Virtual Machine Monitor**. Proceedings of the 2001 USENIX Annual Technical Conference. P. 1 – 14.

SnailDB: Banco de Dados Orientado a Objetos utilizando Prevayler

Giovane Roslindo Kuhn (FURB)
brain@netuno.com.br

Vitor Fernando Pamplona (FURB)
vitor@babaxp.org

Categoria: Banco de Dados.

Linguagem de programação: Java.

Sistema operacional: Qualquer com suporte a JRE 5.0.

Palavras-chave: DDL. DML. JDBC. Persistência. Prevayler. SGBDR. SGBDOO. SQL.

1 Contexto

A programação orientada a objetos já existe há muitos anos, mas somente a partir de 1998 começou a receber fortes investimentos internacionais e evoluções consideráveis nas pesquisas da engenharia. Nos últimos anos notou-se que a maioria das aplicações usa orientação a objetos (OO) para criar telas, regras de negócio e a sua arquitetura, mas continua mantendo os seus dados de maneira relacional, nos bancos de dados relacionais. Para a comunicação entre uma arquitetura OO e base de dados relacional, foram criados vários conceitos, técnicas e *frameworks*, todos eles baseados no conceito de mapeamento objeto/relacional.

A proposta do projeto SnailDB é criar um *framework* que permita que uma aplicação legada, ou já existente, tenha a sua fonte de dados alterada do modelo relacional para o orientado a objeto sem alterar uma única linha de código, como pode ser visto na fig. 1. Para isto, o *framework* permite definir, consultar e manipular coleções de objetos como se fossem tabelas de um banco, inclusive possuindo uma interface de comandos *Structured Query Language* (SQL). Assim, ao executar todos os *scripts* de criação da base de dados no SnailDB, é possível, aos poucos, retirar a parte relacional da aplicação legada, permitindo-a acessar todo o grafo de objetos criado pelo *framework* diretamente.

Figura 1 – Arquitetura em migração

Como conseqüência, a arquitetura do SnailDB também pode ser utilizada para agilizar a criação das consultas em coleções de objetos Java e permite uma manipulação em massa dessas coleções. A linguagem SQL poderá executada sobre *Lists*, *Maps* e outras estruturas existentes na linguagem.

O *framework* inicial tem as mesmas características de um SGBDR[1], isto é, trabalha com dados de forma relacional, via uma interface Java Database Connectivity (JDBC), e grava os dados de maneira orientada a objeto. Ele é capaz de interagir com qualquer aplicação feita em Java, como se fosse um banco de dados relacional comum.

---

1 Sistema Gerenciados de Banco de Dados Relacional

As características desenvolvidas para o SnailDB, em sua primeira versão, são as seguintes:

a) uma camada de persistência dos objetos utilizando a Application Programming Interface (API) de prevalência Prevayler (PREVAYLER TEAM, 2004). O Prevayler é um projeto brasileiro para persistência de objetos Java com extrema facilidade de uso. O grafo dos objetos em memória é persistido de maneira que, ao ser carregado, todas as referências dos objetos, suas dependências e associações continuem intactas;
b) uma linguagem para criação das classes e manipulação de dados em tempo de execução, seguindo as especificações da Data Definition Language (DDL) e Data Manipulation Language (DML) respectivamente. Os comandos implementados são o *CREATE TABLE*, que criará uma classe Java, e o *INSERT INTO*, que irá criar e preencher os objetos definidos pelo *CREATE TABLE*;
c) uma linguagem para consulta de objetos seguindo a especificação da SQL92. O comando *SELECT* deve ser implementado com possibilidade de junções e filtros.

2 Desenvolvimento

Inicialmente foi definida a BNF para a linguagem de criação, manipulação e seleção dos objetos no SGDB. O compilador para a linguagem é gerado com o JavaCC (JAVACC, 2005), um gerador de compiladores que utiliza a BNF definida como entrada.

O comando CREATE TABLE gera, fisicamente, o fonte de uma nova classe no banco. Este fonte é gerado com o auxílio do Velocity (APACHE FOUNDATION, 2005), um gerador de código baseado em *templates* e compilado com o compilador do Eclipse (ECLIPSE FOUNDATION, 2005), ambiente de desenvolvimento utilizado no projeto. Ao gerar a classe, o SnailDB popula uma estrutura de metadados, que é mantida junto com os futuros dados desta classe no Prevayler. Assim, quando a aplicação for reiniciada, a base de dados pode continuar com o mesmo estado da última execução.

As classes são geradas dinamicamente no SGBD, porém o *classloader*[1] padrão do Java não enxerga as classes geradas. Para resolver esta questão, foi implementado um *classloader* que conhece as configurações do SGBD e conseqüentemente a localização física das classes geradas.

O comando *INSERT INTO* cria uma nova instância da uma classe gerada pelo comando *CREATE TABLE*. Os objeto instanciados são persistidos no Prevayler para que posteriormente possam ser manipulados e selecionados pelo usuário.

O comando *SELECT* navega pelo grafo de objetos persistidos no Prevayler, selecionando os objetos que satisfazem as condições definidas no comando. No retorno estes objetos são convertidos para dados relacionais, permitindo o desenvolvedor manipulá-los através do JDBC.

Todos os comandos são implementados com o padrão de projeto *Visitor*, que tem o objetivo de navegar na árvore sintática e executar a respectiva operação. Um dos benefícios deste padrão é o desacoplamento da sintaxe da linguagem da forma com que os comandos são executados, com isso, mudanças na forma de execução dos comandos não afetam a árvore sintática da linguagem.

3 Resultados

Para a validação do *framework* foram desenvolvidos testes unitários simulando a execução de uma aplicação. Nos testes, a definição de classes simples, isto é, sem associações, puderam ser validadas, assim como a inserção e seleção de objetos. Outra implementação para validação foi um terminal de interação com o usuário, onde é possível escrever comandos para o SnailDB executar e então visualizar as respostas retornadas.

---

[1] Mecanismo que o Java utiliza para carregar classes para a memória do programa

4 Referências

APACHE FOUNDATION. **Velocity**. [S.l.], 2005. Disponível em: <http://jakarta.apache.org/velocity/>. Acesso em: 14 out. 2005.

ECLIPSE FOUNDATION. **Eclipse**: main page. [S.l.], 2005. Disponível em: <http://www.eclipse.org/>. Acesso em: 14 out. 2005.

JAVACC. **Java compiler compiler**. [S.l.], 2005. Disponível em: <https://javacc.dev.java.net/>. Acesso em: 14 out. 2005.

PREVAYLER TEAM. **Prevalence skeptical FAQ**. [Curitiba], [2004]. Disponível em: <http://www.prevayler.org/wiki.jsp?topic=PrevalenceSkepticalFAQ>. Acesso em: 14 out. 2005.

Desenvolvimento de jogo 3D utilizando *engine* gráfica e física

Christian Rogério Câmara de Abreu (FURB)
crca.com@gmail.com

Fernando dos Santos (FURB)
fds.com@gmail.com

Categoria: Computação Gráfica

Linguagem de programação: C++

Sistema operacional: Windows XP

Palavras-chave: *High-concept. Engine.* Jogo 3D.

1 Contexto

Este relato foca o desenvolvimento de um jogo para um cenário em três dimensões (3D) aplicando a *engine* de renderização gráfica OGRE (OGRE TEAM, 2005) e de simulação física ODE (SMITH, 2005).

O objetivo principal deste artigo é demonstrar o desenvolvimento com *engine*s gráficas e físicas, elucidando sua utilização para permitir o emprego das mesmas na implementação de jogos.

2 Desenvolvimento

O jogo foi desenvolvido a partir da elaboração de um *high-concept* (ROLLINGS; ADAMS, 2003), que descreve de forma sucinta o enredo e a dinâmica do jogo. A implementação foi realizada utilizando-se Microsoft Visual Studio 6.0 e aplicando a metodologia de Análise Orientada ao Objeto (AOO), onde cada objeto 3D foi representado por uma classe. Na figura 1 é apresentada uma tela do jogo desenvolvido.

Figura 1 - Jogo 3D

A *engine* OGRE foi utilizada para renderizar o cenário e os objetos 3D. Já a *engine* ODE foi utilizada para detectar as colisões dos tiros com os obstáculos.

3 Resultados

Como resultado foi desenvolvido um jogo 3D que permite ao jogador pilotar uma nave espacial num planeta onde estão dispostos obstáculos que devem ser destruídos a partir de tiros disparados pela nave.

A utilização do *high-concept* permitiu agrupar as funcionalidades e os componentes que estariam presentes no jogo. A aplicação da AOO possibilitou a representação dos componentes 3D do jogo de maneira clara e abstrata.

4 Referências

OGRE TEAM. **Object oriented graphics render engine**. Channel Islands, UK, 2005. Disponível em: <http://www.ogre3d.org>. Acesso em: 30 jun. 2004.

ROLLINGS, Andrew; ADAMS, Ernest. **Andrew Rollings and Ernest Adams on game design**. Indianapolis, Indiana : New Riders, 2003. 621 p.

SMITH, Russel. **Open dynamics engine**. Califórnia, 2005. Disponível em: <http://www.ode.org>. Acesso em: 14 out. 2005.

Simulação de robôs aplicando o Algoritmo de Dijkstra e Subida da Montanha

Christian Rogério Câmara de Abreu (FURB)
crca.com@gmail.com

Dr. Oscar Dalfovo (FURB)
dalfovo@inf.furb.br

Categoria: Inteligência Artificial

Linguagem de programação: Java

Sistema operacional: qualquer sistema com a Java Virtual Machine (JVM)

Palavras-chave: TeamBots. Dijkstra. Subida da Montanha.

1 Contexto

Conforme Balch (2000), TeamBots foi desenvolvido pela Universidade Carnegie Mellon. Este ambiente foi implementado na linguagem Java, o qual permite simular a execução de um robô. O software gerado com o TeamBots pode ser instalado na arquitetura de hardware Ninnow, que é um robô que executa os bytecodes do software.

Este trabalho objetiva fazer um robô no ambiente TeamBots que descobra um caminho até um ponto escolhido aplicando o algoritmo Subida da Montanha e o Dijkstra.

2 Desenvolvimento

As ferramentas e técnicas utilizadas foram o algoritmo Subida da Montanha, algoritmo de Dijkstra, linguagem Java e o ambiente TeamBots.

Foram implementados dois robôs no ambiente TeamBots com o objetivo de buscar o melhor caminho num grafo. Um robô aplica o algoritmo de Dijkstra e outro o algoritmo Subida da Montanha. Respectivamente suas execuções estão representadas na figura 1.
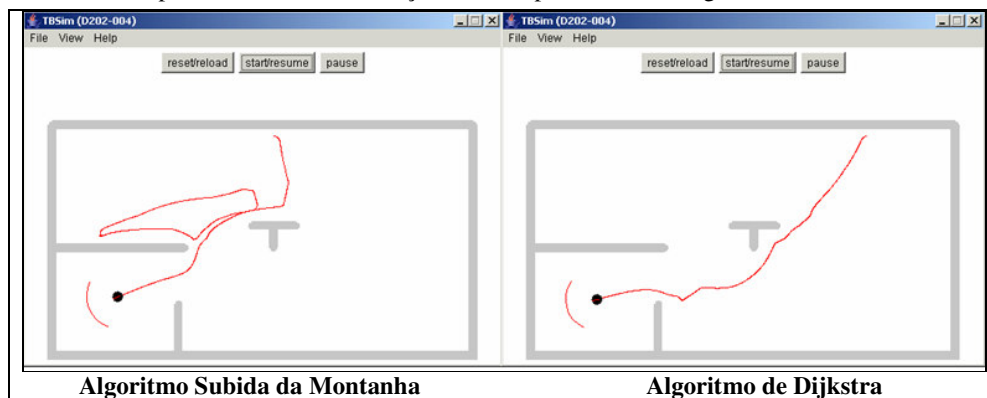
Figura 1 - Execução dos algoritmos

3 Resultados

Foram implementadas duas soluções, uma aplicou o algoritmo Subida da Montanha e a outra o de Dijkstra. Em cada robô foi armazenado um grafo que representa os caminhos possíveis no mapa. Os robôs foram programados para gerar um caminho baseado no grafo, sendo que o ponto inicial do caminho é a posição que está o robô e o ponto final foi escolhido. Os dois robôs obtiveram resultados positivos, pois ambos alcançaram o ponto final.

Entre os dois robôs, o que obteve os melhores resultados foi o que utilizou o algoritmo de Dijkstra, pois gerou e percorreu um caminho menor que o robô que aplicou o algoritmo Subida da Montanha.

4 Referências

BALCH, Tucker. **Teambots**. Atlanta, 2000. Disponível em: <http://www.teambots.org>. Acesso em: 6 dez. 2004.

Índice de Autores